UNIVERSIDAD POLITÉCNICA
DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS
MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE – EUROPEAN MASTER IN SOFTWARE ENGINEERING

Visualization of a Student Predictive Model for 3D Virtual Environments driven to Procedural Training

Master Thesis

KeLi Huang

Madrid, July 2015

This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering.

Master Thesis
Master Universitario en Ingeniería del Software – European Master in Software Engineering
Thesis Title: Visualization of a Student Predictive Model for 3D Virtual Environments Driven to Procedural Training
Thesis no: EMSE-2015-07
July 2015

Author: KeLi Huang
Academic title: Bachelor of Science

University of the presented title: Southwest University of Science and Technology

Supervisor:

Jaime Ramírez Rodríguez
Ph. D. in Computer Science
Universidad Politécnica de Madrid

DLSIIS
ETSI Informáticos
Universidad Politécnica de Madrid

Co-supervisor:

Diego Riofrío Luzcando
BSc. in Computer Science
Escuela Politécnica Nacional del Ecuador

DLSIIS
ETSI Informáticos
Universidad Politécnica de Madrid

ETSI Informáticos

Universidad Politécnica de Madrid

Campus de Montegancedo, s/n

28660 Boadilla del Monte (Madrid)

Spain

Acknowledgements

I would like to thank:

My family: For always supporting me, encouraging me, and for the love they have given me.

My friends: For the advice they've given me, and the best life time they have shared with me in these years.

Jaime Ramirez: For trusted me in personal, directed and helped me during the realization of this work.

Diego Riofrío: For personally trusted me in this work, and for all the knowledge transmitted to me.

Index

ABSTRACT

This document presents the implementation of a Student Behavior Predictor Viewer (SBPV) for a student predictive model. The student predictive model is part of an intelligent tutoring system, and is built from logs of students' behaviors in the "Virtual Laboratory of Agroforestry Biotechnology" implemented in a previous work. The SBPV is a tool for visualizing a 2D graphical representation of the extended automaton associated with any of the clusters of the student predictive model. Apart from visualizing the extended automaton, the SBPV supports the navigation across the automaton by means of desktop devices. More precisely, the SBPV allows user to move through the automaton, to zoom in/out the graphic or to locate a given state. In addition, the SBPV also allows user to modify the default layout of the automaton on the screen by changing the position of the states by means of the mouse. To develop the SBPV, a web application was designed and implemented relying on HTML5, JavaScript and C#.

Chapter 1

1. Introduction

In recent years, educational technologies like educational 3D virtual environments have appeared to meet the needs of some educational problems, such as the impossibility of accessing laboratories with the required equipment; the excessive distance to school/university; the excessive number of students in the classroom; etc. Educational 3D virtual environments improve the student learning by helping them to better understand the concepts and by increasing their learning retention rate.

Intelligent Tutoring Systems (ITS) is an important part of the evolution of the educational technology, because they can give personalized assistance to students in computer based educational systems like interactive simulations.

The Student Behavior Predictor (SBP) is the most concerned component in the ITS architecture proposed by *Diego Riofrío* and *Jaime Ramirez* [1]. This component is able to predict the most probable student's next action using the action records from past students. Within the SBP lies a student predictive model, which is a summarized representation of the past student actions. The project is recording data from students every year, and they have collected a significant amount of action records so far.

The main objective of this work is to develop a tool for visualizing a 2D graphical representation of the student predictive model in SBP. As the model is expressed in form of a big extended automaton, in general, it will not be possible to render the

whole automaton on the screen at once with an acceptable resolution. As a result of this, the tool will also support the proper interaction mechanisms based on desktop devices: to zoom in/out some selected parts of the automaton; to navigate forward or backward; to search some state specified by its name; etc.

The structure of the remaining documents is as follows:

The chapter “Related Work” describes briefly the main technologies used in this project and explains why each of these technologies was adopted in the project.

The chapter “Requirements Specification” is about the project’s initial requirements and it is documented using *IEEE 830-1998* standard.

The chapter “Development of the proposed solution” describes the overall architecture of the application and the detailed design of the solution.

The chapter “Testing phase” shows some of tests performed during the testing of the application by indicating for each test, the expected results and the actual results shown with screenshots.

The final chapter consists of the conclusions of this work and several ideas for the future work that could be derived from this project.

Chapter 2

2. Related Work

The project was originally supposed to be a web application, and was defined to be built as an ASP .NET project, which in turn involves C# and HTML5. Moreover, in order to reach a better performance on the client side of the application, JavaScript was also adopted. Later on, the SVG language was also adopted since it works better with HTML5 than the Canvas element for our purposes.

2.1 ASP .NET MVC

ASP.NET MVC is an implementation of the MVC pattern part of ASP.NET Web Application framework, which is officially supported by Microsoft. It comes from the MonoRail of Castle, and its latest version is ASP.NET MVC 5.1. [2]

ASP.NET MVC Framework is added to ASP.NET by Microsoft as a set of class libraries. This group of class libraries can use the Model-View-Controller design pattern to develop ASP.NET applications. It does not conflict with existing ASP.NET application, therefore both of them can be employed in parallel. ASP.NET MVC Framework is packaged in System.Web.Mvc.dll and uses the ASP.NET Routing to support the action flow and the ability of URL rewriting, which makes it closer to Web development and Web 2.0 features.

As ASP.NET MVC is built-in Visual Studio, an ASP.NET MVC framework project can be easily built in visual studio, developers only need to choose a template, and then Visual Studio will create the corresponding directory structure.

The newest ASP.NET 5 is a cross-age rewrite, all the functions and modules have been split independently to improve decoupling. For these changes, Microsoft almost rewrote all the .NET Framework again, forming the so-called .NET Core thing. In the .NET Core everything is configurable, including Session, MVC and other features, and all the configurable features can be downloaded at Nuget.

Why MVC

A big change in ASP.NET 5 is that there's no more Web Forms. The reasons of this are obvious, because Web Forms are having problems because of its design defects [3]:

1 It only provides a view based solution where an action based solution is required, which leads to an unfitting and illogical architecture.
2 As side effects of a bad architecture, it has a very tight coupling.
3 As a result of the tight coupling between view and code behind, even the response type is fixed to HTML in webform by default
4 Webform is a view based architecture, therefore each view determines which model must be connected with it making the view less flexible and also involving view in complex decision making, which clearly violates SRP of the SOLID principles.
5 The behind code a webform is a very typical heavy and bulky weight partial class, which cannot be instantiated in simple C# code straightforward.

According to MVC pattern, the basic idea is a clear division of work, with Controller as the core part. In the development process Models can be done at the first, then Controllers, and Views at last, by using the demo view at the beginning, and finally replacing it by the real view. This pattern has changed the development

thinking from view-centric to data-centric. The most important advantage is that each part of this model can be reused in a flexible way to meet a variety of network needs now, such as the Web and mobile devices. Moreover, other frameworks, in ideological terms are basically using this pattern too, which ultimately proved by time, has become the standard way of thinking and developing. The visual-oriented development concept promoted initially by Microsoft, has gradually retreated its light of the past.

2.2 C#

As Microsoft has no plans to support Visual Basic in ASP.NET 5 in terms of compilation, project templates, and other tools, C# became the best approach for ASP.NET projects [4]. C# is a secure, stable, simple, elegant, object-oriented programming language derived from C and C ++. It inherits the power of C and C ++, while removing some of the complexity in their characteristics (e.g. no macros, and does not allow multiple inheritance). C# combines the simple visual operation of Visual Basic and the high efficiency of C ++, which together with its strong operational capability, elegant syntax style, innovative language features and convenient support for component-oriented made it the first programming language choice for .NET development.

On the other hand, to ease interoperability with the Student Model (see section 6), C# was chosen to be used in the part of building a MVC (Model-View-Controller) structure, and to prepare web pages and back-end data in the server side.

2.3 HTML5

HTML5 is the fifth major revision of HTML, a core technology markup language of the Internet used for structuring and presenting content in the World Wide Web [5]. When a browser requests an ASP.NET file, ASP.NET engine reads the file, compiles and executes scripts in the file, and then return results in plain HTML to the browser.

HTML5 is a technology recommended by the W3C, and its development is the result of the cooperation of hundreds of companies. The biggest benefits of this technology is that it is an open technology. In other words, every open standard of HTML5 can find its root causes on the W3C's database. On the other hand, any standard adopted by the W3C will be implemented by every browser or platform.

From the perspective of this work, there are several reasons to choose HTML5:

- OFFLINE & STORAGE: Web Apps can start faster and work even if there is no internet connection, thanks to the HTML5 App Cache, as well as the Local Storage, Indexed DB, and the File API specifications.
- CONNECTIVITY : More efficient connectivity means more real-time chats, faster games, and better communication. Web Sockets and Server-Sent Events are pushing (pun intended) data between client and server more efficiently than ever before.
- 3D, GRAPHICS & EFFECTS: Between SVG, Canvas, WebGL, and CSS3 3D features, you're sure to amaze your users with stunning visuals natively rendered in the browser.

2.4 JavaScript

JavaScript [6] is an Internet scripting language that has been widely used in Web application development, typically used to add a wide range of dynamic functions for web pages, and to provide users with a smoother experience as using browsers. JavaScript is usually embedded in HTML to achieve its functions. It can be embedded directly into HTML pages, but written in a separate js file, which is conducive for the separation of structure and behavior.

JavaScript scripting language has the following features:

- JavaScript is an interpreted scripting language, unlike C, C ++ or other languages that will compile before run; instead, JavaScript is interpreted line by line in the process of running the program.
- JavaScript is a scripting language based on objects, it can not only create new objects, but also use existing objects.
- JavaScript language is using weak variable typed, which does not make strict requirements on data type. It is based on the basic statements and control of Java, and its design is simple and compact.
- JavaScript is an event-driven scripting language, which can respond to user input without going through the Web server
- JavaScript does not rely on the operating system, since only needs browser support. Therefore, after a JavaScript program is written, it can be taken to run on any machine whenever the browser on the machine supports JavaScript, which is very likely because JavaScript is now supported by most browsers.

In order to achieve goals of user interaction, JavaScript would be in charge of handling user actions in client part of the application, and also the communication with server back-end. JSON or XML are typically used as data formats in the communication between client and server.

2.5 SVG

Scalable Vector Graphics (SVG) [7] is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. As the matter of fact, SVG is an XML file, therefore SVG also inherited the advantages of XML such as openness, portability and interactivity. Today, almost all mainstream browsers support SVG.

In most cases, we use raster graphics and tools to create them, because they are easier to get and use. However, there is an irreplaceable advantage of vector graphics that raster graphics doesn't have: No matter how one zooms a picture, it will not be distorted. This is especially relevant when we need to have accurate measurements, and ability to zoom in to see the graphic details.

At the beginning of the design of the application, the new HTML5 element canvas seemed to be the best option for painting images, but after a deeper study, we realized that SVG fits better to our needs. Canvas is rendered pixel by pixel, which means that once the graphic is drawn, it will not keep the browser's attention. If the graphic is changed, then the whole scene needs to be redrawn. In contrast, SVG is based on XML, which means that each element of the SVG Document Object Model (DOM) is available to be handled as an independent object and it can be attached with JavaScript event handlers. Hence, if any property of the SVG object changes, the browser can automatically redraw only the graphic of this object.

2.6 IDE

There were two options we were considering to be the IDE:

- Xamarin
- Microsoft Visual Studio

Xamarin[8] was founded in 2011, designed to make mobile development becomes incredibly quick and easy. There are many known open source community developers participate in the creation of Xamarin, Xamarin is also the leader of the open source Mono project -- cross-platform implementation of C# and .NET framework. Xamarin Studio is a C# integrated development environment for cross-platform development, it's tightly integrated with iOS and Android SDK and it also integrates Git and Subversion. Xamarin Studio can perfectly support Visual

Studio on Windows and Mac.

Microsoft Visual Studio is the most popular integrated development environment for Windows platform applications nowadays, as the IDE of Microsoft itself, it always integrates the newest version of .NET framework, and also the most complete scheme for .NET framework.

Troubles with the Web server and SVG

Firstly we tried to use the ASP.NET MVC framework in Xamarin, so we created a default project with Xamarin Studio and started working with it. However, during the implementation, after implementing the Student Model in the project, a problem occurred with Xamarin, and no matter how hard we tried, the SVG document could be not parsed by the browser. After several tests, we located the issue in the Xamarin server, because the browser was not receiving the right format of files from the server. After examining the Xamarin features, we found out that the problem was that current version of Xamarin does not support the selection of the web server during debugging, and its default internal server does not support the transfer of SVG file formation.

This forced us to use the ASP.NET MVC framework attached in Visual Studio. Then, we created a default project with Visual Studio using IIS Express as the debugging server.

2.7 Debugging

The back-end debugging was done with Visual Studio.

The front-end debugging was using Element Inspection from browser. We used chrome as the default debugging browser, since it has become the most popular browser on the world.

2.8 Student Action Prediction Model

The Student Action Prediction Model **[1]** is a predictive student action model, which uses student logs generated by a 3D virtual environment for procedural training to elaborate summarized information. These student logs are read from a student ontology developed in a previous work. The action prediction based on this model helps an intelligent tutoring system to generate students' feedback proactively, i.e., it can show hints to students before they perform an action or make a mistake.

2.8.1 ITS Architecture Proposal

For a better understanding of the model, we are going to explain the ITS architecture. The ITS architecture proposal is inspired on MAEVIF architecture [9] [15] and is depicted in Figure 1. The principal modules of this architecture are the following ones: Communication Module; Student Module; Expert Module; World Module; and Tutoring Module.

Tutoring Module works as the core of the ITS, and it is the one that performs most of the computational logic of the tutoring task. This module is responsible of validating the students' actions by using the information about this action contained in the Expert Module. In addition, it is also responsible of showing the hints or error messages that are more pedagogically appropriate in each moment.

This module has the following components:

- Tutoring Coordinator
    - Student Feedback Generator

- Student Behavior Predictor

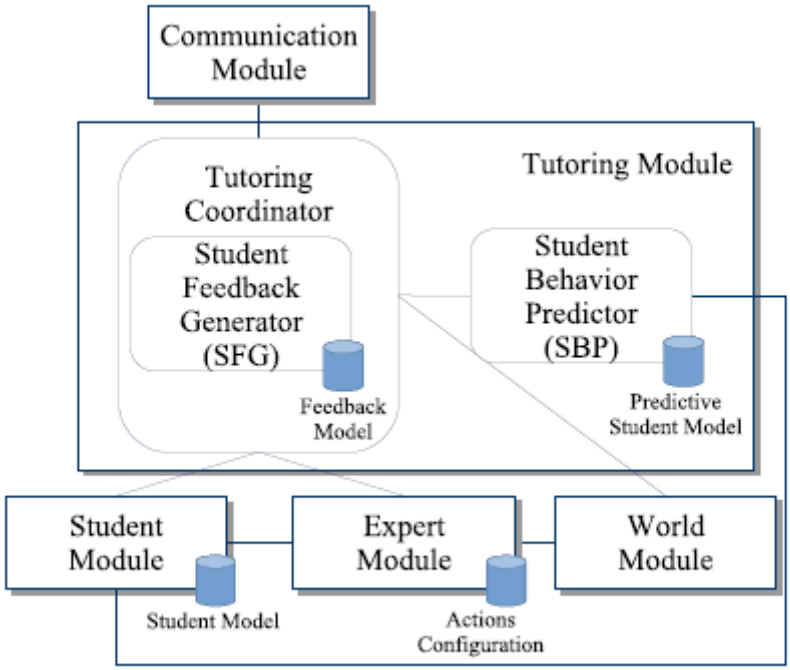


Figure 1: ITS architecture proposed

Tutoring Coordinator is responsible of the validation of students' actions. So, it validates whether an incoming action attempt from the Communication Module is correct, and then informs of the result of this validation to the Student Feedback Generator (SFG), the Student Behavior Predictor (SBP) and the Student Module.

The SFG is responsible of providing the Tutoring Coordinator the most appropriate tutoring feedback. To this end, SFG relies on Feedback Model, which defines a set of rules that takes this decision considering what infers the Student Module about the student knowledge; the most probable next student's action found by the SBP; and the confidence degree of the information supplied by the Student Module and the SBP. So, if SFG does not trust neither Student Model nor SBP, then it will mean that the SFG does not have enough information on the student under supervision, and therefore the most reasonable alternative of the SFG will be to provide a generic tutoring feedback following the approach implemented in the reactive tutor. Otherwise, the SFG will be able to provide a more personalized

tutoring feedback.

The SBP is the component that attempts to predict the next most probable action based on the action records of past students. For this, it relies on the Predictive Student Model, which comprises these action records summarized. Once that most probable action is found, it is delivered to the SFG along with the confidence of this prediction.

For a better understanding of how SBP works, here is a brief class diagram of Student Behavior Predictor (Figure 2):

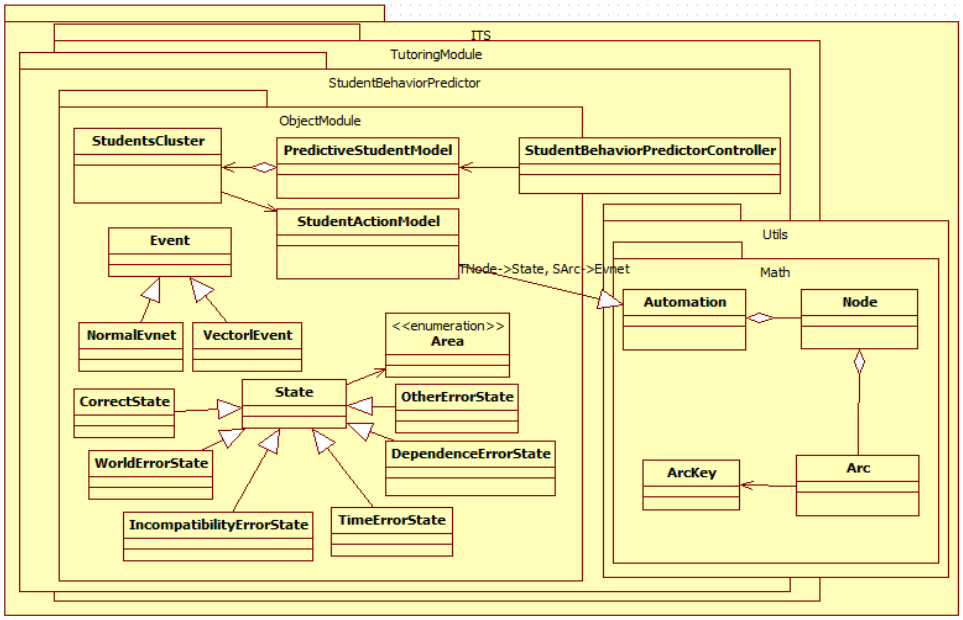


Figure 2 - Student Behavior Predictor class diagram

The detailed class diagram will be shown in Annex 1.

2.8.2 Predictive Student Model

The idea of building this model arose from our experiences evaluating the 3D virtual lab of biotechnology and the reactive tutor mentioned above [22]. During the model design, it was necessary to observe the behavior of students in the virtual world following the ethnographic method. Subsequently, as recommended by

Mostow et al. [21], the logs generated by the reactive tutor were analyzed by hand to identify interesting phenomena.

As mentioned previously, this model is part of the component Student Behavior Predictor of the Tutoring Module. It contains summarized data from historical registries of actions made by past students, and it is used by this component to obtain the next most probable student's action. The model (see Figure 3) consists of

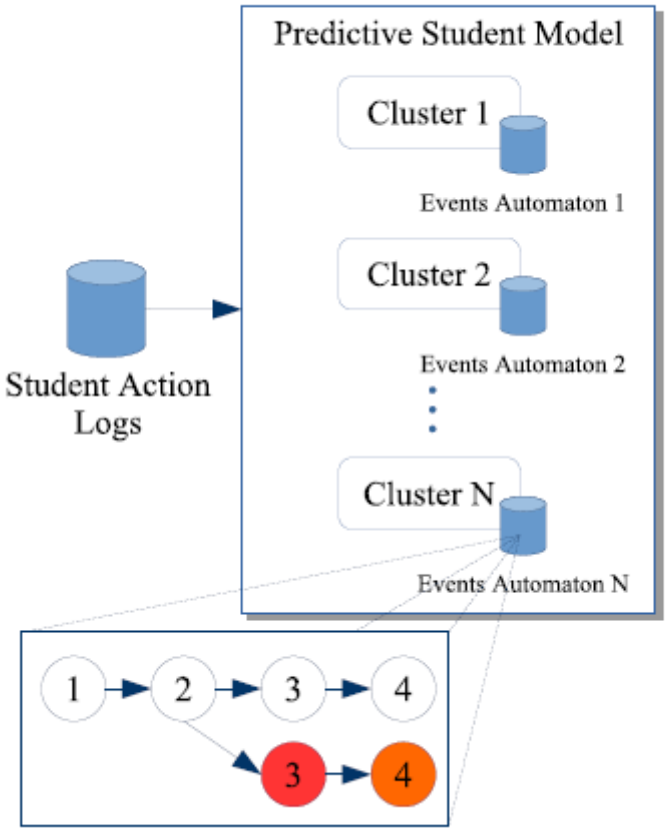


Figure 3: Predictive Model

several clusters of students where each of them contains an extended automaton, detailed in section 4.1. These clusters help to provide automatic tutoring adapted to each type of student. For example, if the student is not committing many errors, it is more probable that his/her next action will not be an error. However, for a student who has failed more times, it will happen the opposite.

2.8.3 Extended Automaton Definition

This automaton consists of states (represented by circles) and transitions (represented as arrows) as shown in figure 3. Furthermore, the states are grouped into three zones: Correct Flow, Irrelevant Errors and Relevant Errors Zone.

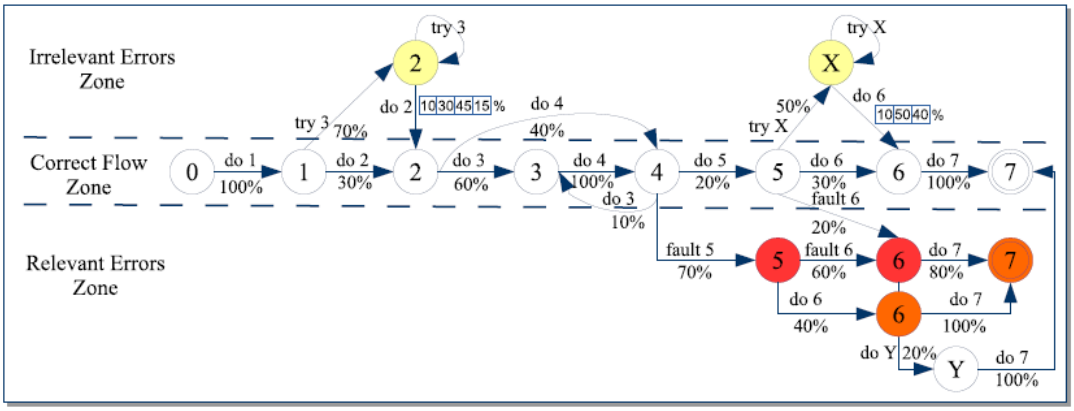


Figure 4: Example of an extended automaton

2.8.4 Correct Flow Zone

In this area are the events that represent the valid sequence of actions for an exercise, which logically ends up with a final satisfactory state. These states are graphically represented by white circles (see figure 4).

2.8.5 Irrelevant Errors Zone

This zone groups states derived from error events that do not influence in the final result. These error events are associated with action attempts blocked by the tutor, because, according to the tutoring strategy, their execution cannot be allowed. They are also defined as blocking errors [22], because the exercise cannot continue if they occur until a valid action is executed. These states are graphically represented by a yellow circles.

2.8.6 Relevant Errors Zone

This area encompasses states derived from error events that actually influence in

the final result, i.e. if an event of this type occurs the final result will be wrong unless a repairing action is done.

Chapter 3

3. Requirements Specification

3.1 Introduction

3.1.1 Purpose

The purpose of this document is to state the requirements for the Student Model Predictor Viewer.

3.1.2 Product Scope

The product is a software application that is accessed from a web browser and used to show a graphic of the Student Predictive Model, and supports simple interactions with the user.

3.1.3 Glossary

SBP: Student Behavior Predictor

ITS: Intelligent tutoring system

3.1.4 References

- IEEE 830-1998

3.1.5 Overview

This section is laid out in a modified IEEE830-1998 style. The following information is in this section:

- Section 2: General description of application, dependency analysis, and enterprise requirements.
- Section 3: Functional & Non-functional requirements. Note that all non-functional requirements are grouped into one category, and not distributed among categories, as in IEEE830. Section 3 also covers use cases and deleted requirements.

3.2 General Description

This application is defined to be part of the Student Behavior Predictor system, which is a project working on collecting big data of students' behaviors from a Virtual Biotechnology Laboratory for training. The intention of the application is to support users to view the extended automaton explained in section 6.3 from the structure level to the detail level in a graphical way.

3.2.1 Product Perspective

The Student Predictor Viewer is a graphical data access application; however, it will require users to have access to a web browser on their workstation computer, and to Student Behavior Predictor System. This means that the users of the system do not need to spend any money in the Student Behavior Predictor System to get the most from it as any Windows based PC brings installed a web browser, and any non-Windows machine can use Chrome or other freeware browsers.

3.2.2 System interfaces

As stated in section 2.1 the Student Predictor Viewer is a data access application,

relying on the Student Behavior Predictor System. This means the system will require interfaces with the Student Behavior Predictor System (through a DLL library). The System interfaces required by the system server are the following:

- Network interface to a network with an internet connection

- Connection to the Student Model in Student Behavior Predictor System which containing students' behaviors data

3.2.2.1 User interfaces

The application is a web-enabled system, meaning that all the user interactions are done through a web browser.

3.2.2.2 Hardware interfaces

There are no hardware interfaces to this system.

3.2.2.3 Software interfaces

There are no hardware interfaces to this system.

3.2.3 Product Functions

3.2.3.1 Show extended automaton of the student model in an appropriate form

3.2.3.2 Show states and events details by clicking on them

3.2.3.3 Allow user to browse the graphic through actions such as move, zoom, flip

3.2.3.4 Search a specific state or event

3.2.3.5 Move a state and their related events.

3.2.4 User Characteristics

Professors and researchers who are using Student Behavior Predictor Model as part of an ITS system. In the future, teachers will use also the viewer to improve their

teaching and the tutoring feedback provided by an ITS.

3.2.5 Constraints

The Student Predictor Model is introduced through a Dynamic Link Library (DLL), which means every time the code of this model is changed, the DLL shall be also updated.

3.2.6 Assumptions and Dependencies

The server program will run with Student Behavior Predictor. The client application will run under a web browser with at least support for HTML5, and will be connected with Internet.

3.3 Specific Requirements

3.3.1 External Interfaces

Browser: All user interfacing is done through a web UI. The external browser needs to support at least HTML5 to use the application.

3.3.2 Functions

3.3.2.1 Use Case: 1 Browsing Student Model

CHARACTERISTIC INFORMATION

Goal in Context: Get an initial graphic of the extended automaton, and browse the graphic by doing actions such as dragging it with mouse, using mouse wheel to zoom in/out, or clicking on the next page button.

Scope: Student behavior predictor viewer

Level: Primary task

Preconditions: None

Success End Condition: The graphic is shown appropriately without graphical overlap or any non-understandable information

Minimal Guarantee: A graphic can be shown, and actions can be done correctly.

Primary Actor: The graphic controller

Trigger: The user select a certain cluster from a certain model to be shown

---

MAIN SUCCESS SCENARIO

1. Application shows a selection of all models.
2. User selects one of the model, application shows a selection of all clusters in the model.
3. User selects one of the clusters, and confirm the selection, application shows the initial graphic of the cluster,
4. User uses the mouse to drag the graphic to view non visible parts of the extended automaton.
5. User uses mouse wheel to zoom in/out, and so the application shows a more/less detail view of the graphic.

EXTENSIONS

3a, 4a, 5a User select another model:

3a1, 4a1, 5a1 application goes to step 2

4b, 5b User select another cluster:

4b1, 5b1 application goes to step 3

4c, 5c User click the confirm button again:

4c1, 5c1 application shows the initial graphic

3.3.2.2 Use Case: 2 Show states/events detail

CHARACTERISTIC INFORMATION

Goal in Context: User clicks on one of the states to get state detail while viewing the graphic

Scope: Student behavior predictor viewer

Level: Primary task

Preconditions: None

Success End Condition: The viewer shows a message box which contains state detail refers to the state

Minimal Guarantee: The detail can be shown, and when user does a click out of the massage box, the box will close

Primary Actor: The graphic controller

Trigger: The user click on a state

MAIN SUCCESS SCENARIO

1. Application shows the graphic of a cluster.
2. User use the mouse to click on one of the state
3. Application shows massage box with state detail
4. User click out of the massage box, the massage box disappears.

EXTENSIONS

3a User drag the graphic while the massage box is shown:

3a1 The message box shall be kept visible with the same information while the user is dragging the graphic.

3.3.2.3 Use Case: 3 Search a certain event in a graphic

CHARACTERISTIC INFORMATION

Goal in Context: User clicks in the search box while viewing the graphic and enters a text to search a certain state. The matching criterion is implemented in the Student Behavior Predictor Model.

Scope: Student behavior predictor viewer

Level: Primary task

Preconditions: None

Success End Condition: User inputs some text and clicks the button, and the application goes to the state.

Minimal Guarantee: The viewer will show the state and all the nodes and events around it, or otherwise inform the user if there's no match found

Primary Actor: The graphic controller

Trigger: The user input text into the search box

MAIN SUCCESS SCENARIO

1. Application shows the graphic of a cluster.
2. User click on the search box and input some text
3. Application shows a list of events that match the user input
4. User can chose one of the event and click the button to confirm
5. Application show the event with its context nodes and events

---

EXTENSIONS

3a If there is no match for the user input:

3a1 Application shall inform the user that there's no match found

4a User click the button without choosing from the list:

4a1 Application will inform that there's no match found.

3.3.2.4 Use Case: 4 Change states' positions

CHARACTERISTIC INFORMATION

Goal in Context: User clicks down a state and drag the state to a desired position within the graphics area.

Scope: Student behavior predictor viewer

Level: Primary task

Preconditions: None

Success End Condition: When the user drags a state, the state and all the events related to it move simultaneously following the mouse.

Minimal Guarantee: The state is dragable, and responses correctly to the mouse action.

Primary Actor: The graphic controller

Trigger: The user clicks down a state.

MAIN SUCCESS SCENARIO

1. Application shows the graphic of a cluster.
2. User clicks down a state and starts dragging it.
3. The state and all the events related to it move simultaneously following the mouse.
4. When the user clicks up, the state and the events related to it stop moving.

EXTENSIONS

2a If the user mouse goes out of the graphic area:

2a1 The state and events will stop moving until the mouse gets back to the graphic area.

3.3.2.5 Nonfunctional Requirements

3.3.2.5.1 Performance

The system should provide an appropriate level of performance:

* The elapsed time between a navigation event and the complete refresh of the

graphic should be less than 1 second; and

* the elapsed time of a search of a certain state until the result is shown on the screen should be less than 1 second.

3.3.2.5.2 Usability

As in the future this application will be at disposal of teachers, it should be enough easy to use so that a teacher without computer programming background can use it.

Chapter 4

4. Development of the proposed solution

In this section we will describe the architecture and detailed design of the application.

4.1 Application Architecture

As defined in the specification requirements, the Student Predictor Viewer is a web application that will connect with an existing model. As the model is written in C#,

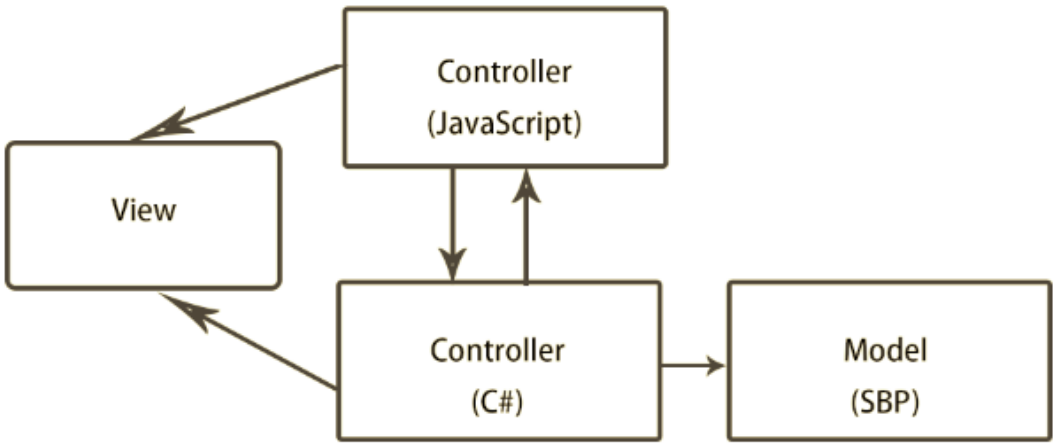


Figure 5 - ASP.NET MVC Directory Structure

then ASP.NET MVC comes to be the most appropriate approach for this purpose. Then we decided to structure the application following ASP.NET MVC framework.

The structure of the application was defined as Figure 5:

An ASP.NET MVC project can be easily built in Visual Studio; its detailed structure is shown in Annexes 2.

4.1.1 Model

The so-called model can be understood as the data, and this data may be data corresponding to a database table. If such data are only properties with no actions associated with them, these data are often referred as entities. Besides the data, model initially also includes interfaces, whose intention is to provide interface standards in order to control the entities, and then provides the carrier of implementation, usually called Mock class. For convenience, a variety of factories may also be created in the Model, which simplifies objects creation.

In the Student Predictor Viewer, as mentioned before, the existing Student Behavior Predictor Model provides a function to create the student models as well as functions to access the student models. The Student Predictor Viewer uses the function to create all models when the server part is starting up. All student models are created in local memory.

4.1.2 Controller

The controller part is a little different from the original ASP.NET MVC framework. As we are using SVG with JavaScript for showing the graphic on browsers, the controller will be divided in two parts, one encoded in C# and another in JavaScript:

4.1.2.1 C# Part

C# is the base language of the entire system, managing from APP_Start to action response, and its part in the controller takes care of the following tasks:

- Choosing the view, which would be only one view so far according to the software requirements, but maybe more than one in the future.
- Calling the functions exposed by the Student Behavior Predictor Model to extract data from the student models.
- Preparing the data into JSON formation and rising query handlers at the back-end.

4.1.2.2 JavaScript Part

JavaScript (JS) as a strong support for web applications is widely used nowadays. More than ten thousand lines of JS code in various web sites has become very common. In the Student Predictor viewer, the JS part is mainly in charge of the user interaction part of controller, i.e., it will take care of the user's actions on the browser. The JS part will perform the following tasks:

- Control SVG graphic: JS queries the data from back-end, stores it at the local storage of the browser, and generates the SVG graphic from this data. It also is responsible of changing the graphic if necessary.
- Reacting to user's actions on graphics: For the purpose of better performance, JS is responsible of reacting directly to actions including dragging, mouse wheeling, etc.
- Query for more data: JS calculates the current state of the viewer and decide if it's necessary to retrieve more data, and if it is, it determines the fragment of data that is needed, and queries the back-end using JSON to obtain that fragment through the C# part.
- Store the incoming data: JS creates and manages a data model to store the incoming data from the Model.

4.1.3 View

In ASP.NET approach, the view is defined in .cshtml files, but to meet the requirements of the project, the view is needed to be dynamically managed by JavaScript during the user session. Hence, the View is also divided into two parts: basic page and SVG element.

The basic page is defined in .cshtml files, and is returned to the user as a “View”. This part contains:

- Page structure
- Selection of Student Model and Cluster

Another part of the View is the SVG element, which is generated and managed by JavaScript. It is the most important part of visual graphic, including the graphical definition of States and Events.

4.2 Detailed Design

4.2.1 Design of the Model

4.2.1.1 Class Diagram

This part mainly includes one class: MapService.

The figure 6 shows the structure of MapService, and how Global class calls it.

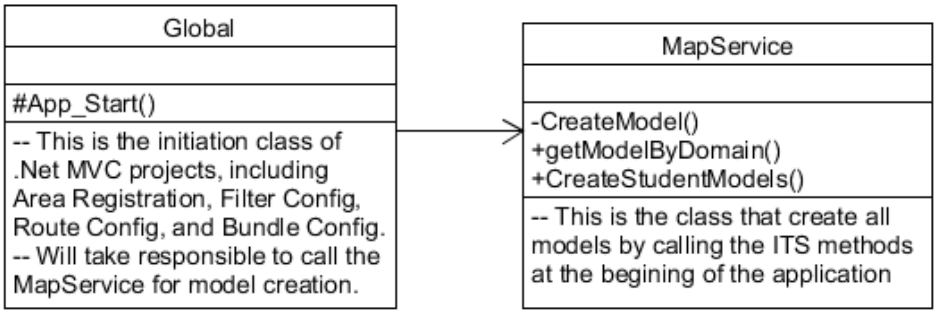


Figure 6 - Structure of MapService with Global

Figure 7 is the runtime sequence diagram of the call of App_Start and the creation of Student Models, showing the cooperation between each component.

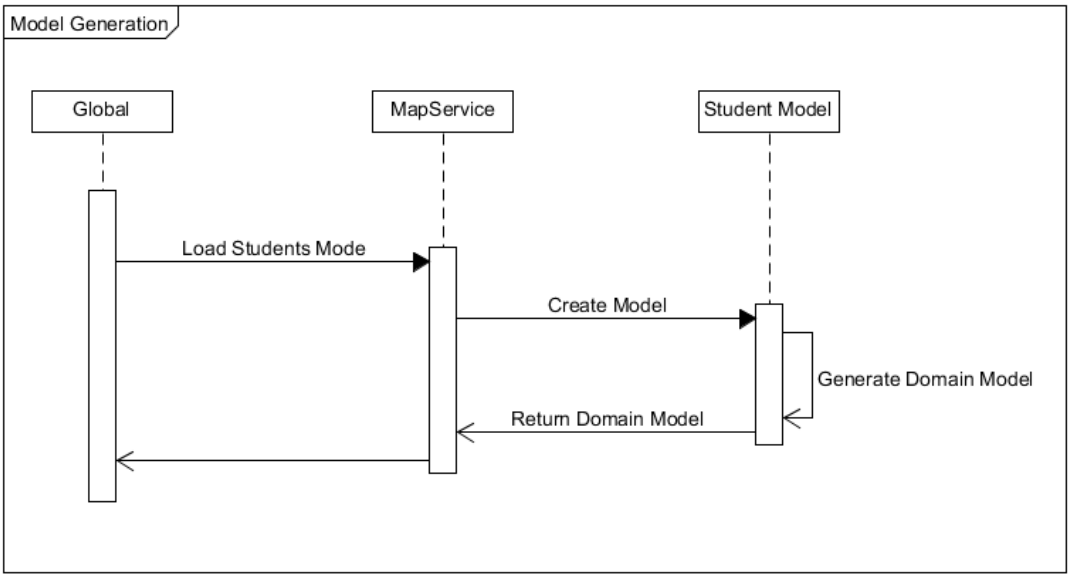


Figure 7 - Model Generation sequence diagram

4.2.1.2 Creating Model

4.2.1.2.1 MapService

This class creates the model using the function *CreateModel,* which is triggered from another called *LoadStudentModels.*

CreateModel: Creates the instance of an specific domain, gets its student logs and passes these logs to the *StudentBehaviorPredictorControl,* which generates the prediction model.

LoadStudentModels: Reading the the configurations, then call *CreateModel* for each domain.

4.2.1.2.2 App_Start

To load the models at the start of the application, the class *Global* was modified as shown in Figure 8.

```
public class MvcApplication : System.Web.HttpApplication
{
    0 references | kelihuang, 11 days ago | 2 changes
    protected void Application_Start()
    {
        ...
        MapService.LoadStudentModels();
    }
}
```

Figure 8 - Global.asax.cs

The process would take a while, but since it would only run once when the server starts, it is considered acceptable.

There were also some configurations modified in importing Student Model, the detailed process of this is attached in Annex 3.

4.2.2 Design of the C# controller

4.2.2.1 Class Diagram

Figure 9 shows the class diagram of MapController, which is the C# controller in the project.

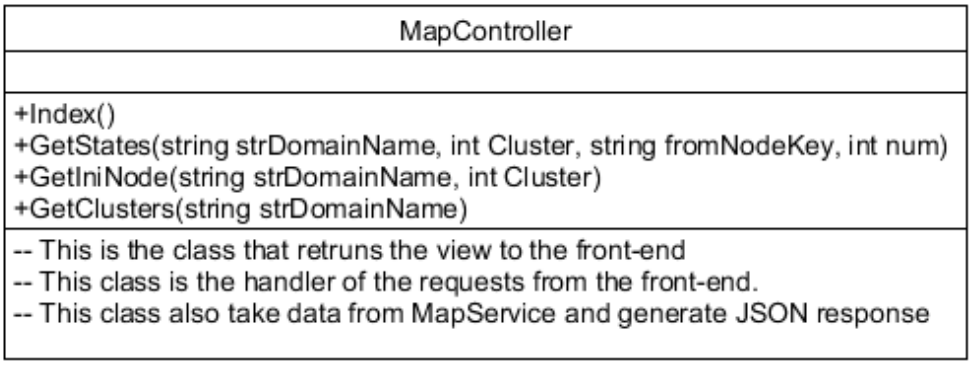


Figure 9 – MapController class diagram

Figure 10 is the runtime sequence diagram of the requests that takes place between the JavaScript controller and C# controller, and which shows the cooperation between the two parts.

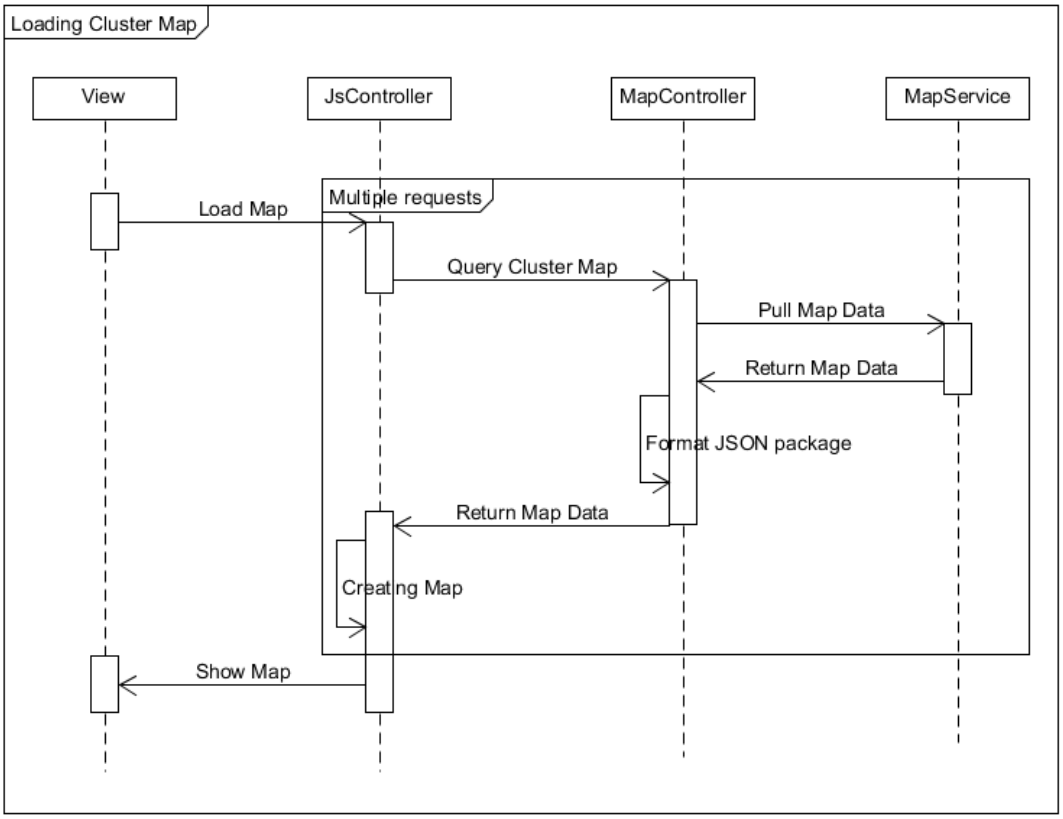


Figure 10 - Loading Cluster Map sequence diagram

4.2.2.2 Actions

To handle back-end actions, a class called *MapController* was created that implements the controller. This class contains three actions*: GetClusters, GetIniNode,* and *GetStates*.

- **GetClusters**: Counts the number of clusters in a domain. This action is called when the user selects a domain. It returns the clusters using JSON format.
- **GetIniNode**: This action is called when the user chooses a certain cluster and loads the map. The action returns an *iniNode* class with a *totalStudent* number in JSON format.
- **GetStates:** This action takes a *nodeKey* as input. Returns all the *Arcs* leaving from the given *Node*. The result is also in JSON format.

4.2.2.3 JSON data formatting

The objects in the built model could not be simply serialized into JSON format, so we decided to choose only the useful information in the objects, and pack it into JSON format by our own. This is done in actions mentioned above.

According to the requirements, for the events (arcs), the JSON data was built with two NodeKeys that indicate the source Node and the target Node. Moreover, JSON data also includes the Frequency attribute, which for Vector events is an array and for Normal events is an integer. Besides these information, we also added an attribute Element, which is a string indicating the type of the event.

For the states, the JSON data was built its basic information: Key, Area, Frequency, name, error associated, etc. We also added the attribute Element with two elements: Column and State Type. Column is an integer that will used to help to locate the state, and State Type indicates the type of this state.

As iniNode is considered a special node, its JSON data was built slightly different including Key, Area, Frequency, Element, and another attribute called Total Students. Total Students is an integer indicating the total number of students in the current cluster map, and will be used for the calculation of Frequencies.

The detailed JSON format will be shown in annex 4.

4.2.3 Design of the JavaScript controller

This part is the main controller of this application so far. It should perform the following tasks:

- Generate map graphic
- Handle user actions

4.2.3.1 Class Diagram

JavaScript Controller

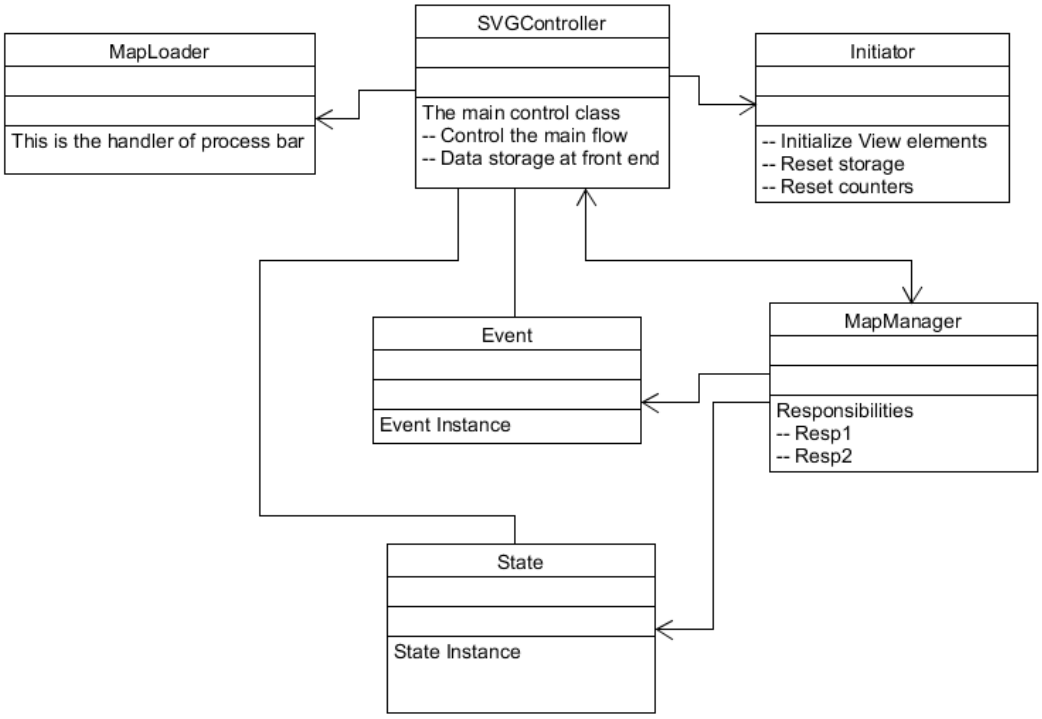


Figure 11 - JavaScript controller class diagram

Figure 11 describes the class structure in JavaScript controller part, the SVGController is the main class that controls the flow, and the others are support classes. Details of each class in the controller are described below:

Dictionary

Dictionary is a support class that implements a Dictionary in JavaScript; its structure is shown in Figure 12.

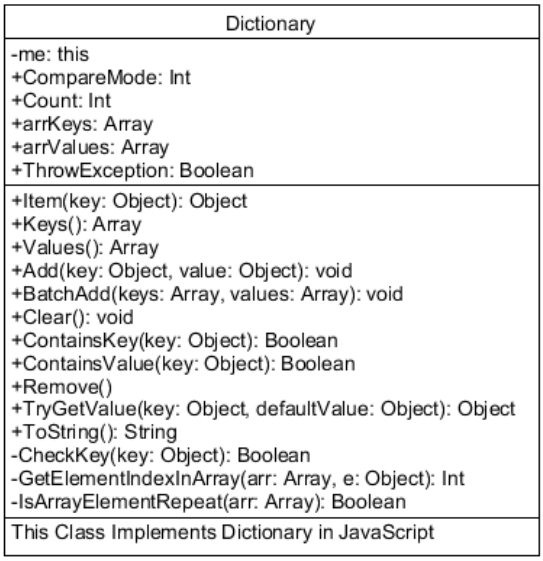


Figure 12 - Dictionary Class Detail

Event

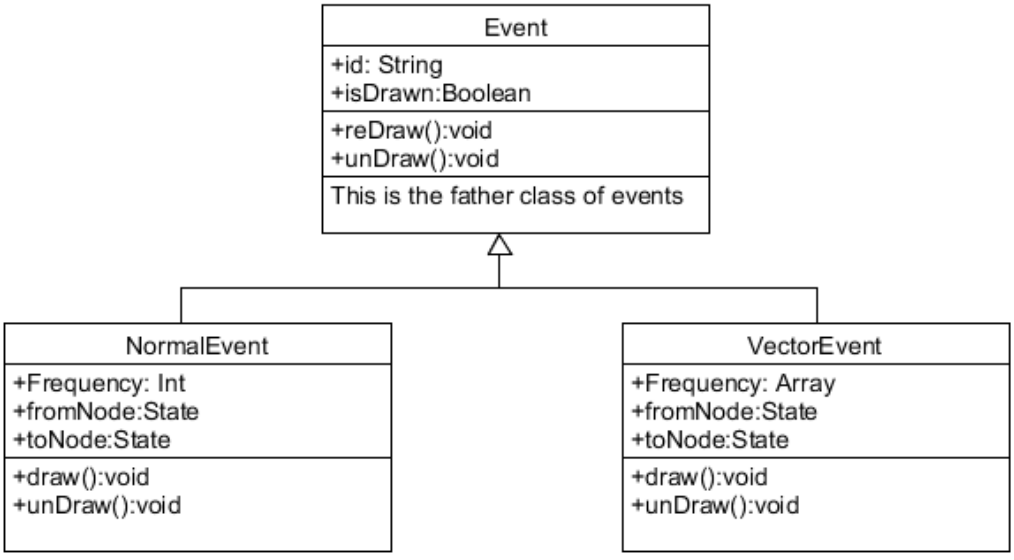


Figure 13 - Event hierarchy

The events are modeled by means of a hierarchy of classes, which consists of an abstract parent class *Event* and two subclasses *NormalEvent* and *VectorEvent*. The structure of these classes is shown in figure 13.

SVGController

The SVGController is the main controller in the front-end that takes the responsibility of map data storage, data instance creation, and handling user actions on graphic. Figure 14 describes the structure of class SVGController.

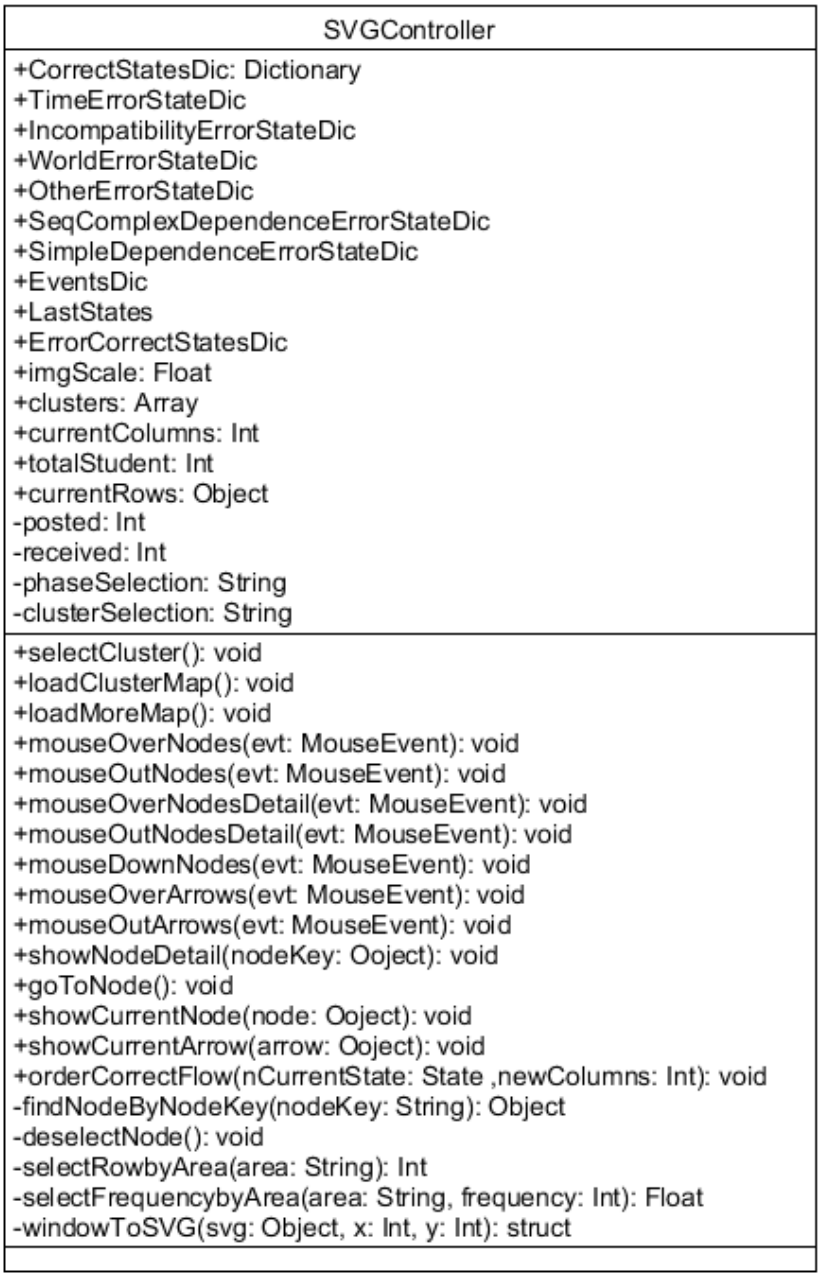


Figure 14 - SVGController Class Detail

State

The states are modeled by means of a hierarchy of classes, which consists of an abstract parent class, which implements the common functions and attributes of all the states, and some subclasses that implements its own *draw()* and *showDetail()* functions. The detailed structure of this hierarchy is shown in figure 15.

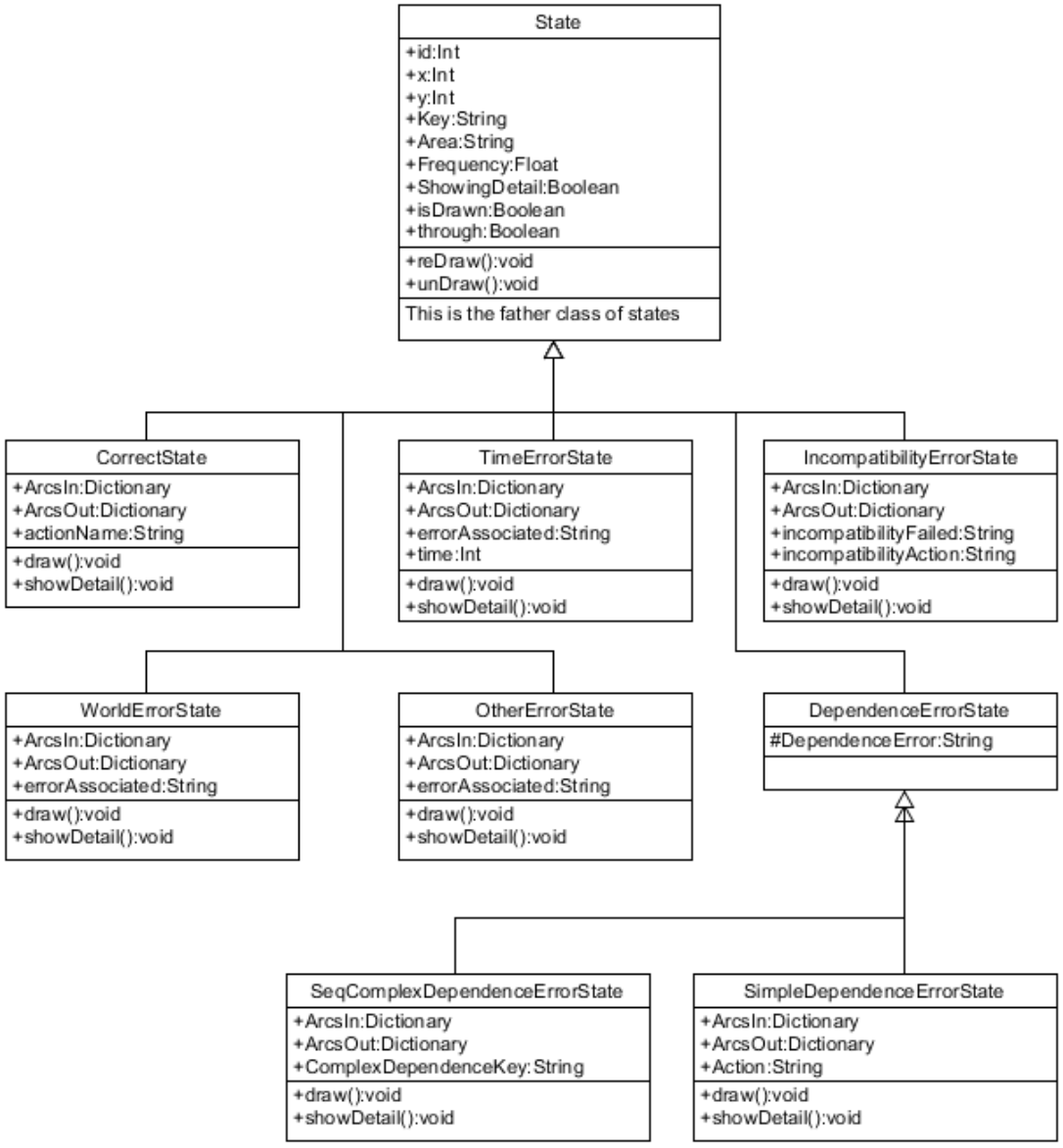


Figure 15 - State hierarchy

MapLoader

MapLoader is a support class to handle the processing bar. Figure 16 shows its structure.

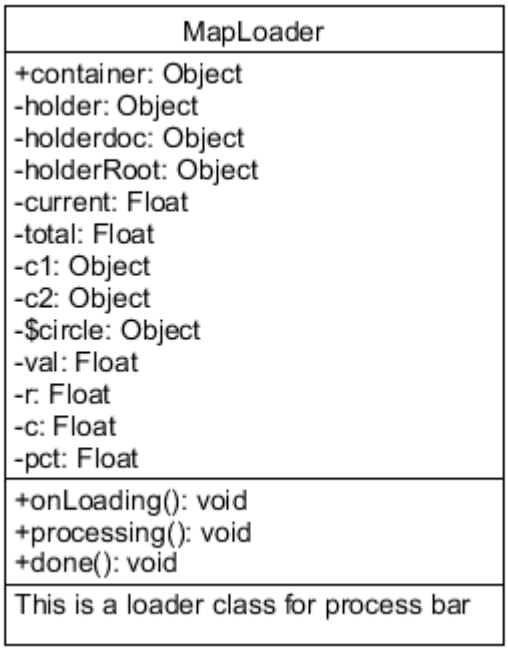


Figure 16 - MapLoader Class Detail

Initiator

The initiator is a support class that is called after loading the map data; it is responsible of initiating the SVG graphic, including the reset of all the data cached. The structure of this class is shown in Figure 17.

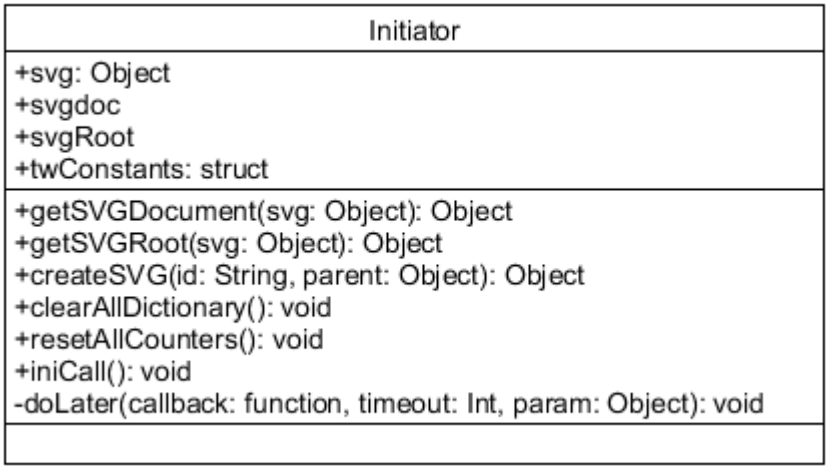


Figure 17 - Initiator Class Detail

MapManager

This class is the map manager at the front-end, and it is responsible for current map detecting and managing. Figure 18 describes the structure of MapManager.

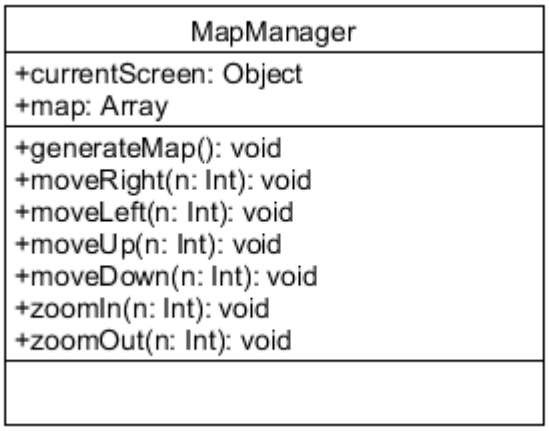


Figure 18 - MapManager Class Detail

Following the workflow of the front-end, we will describe the controller from the point of view of data preparing, initial call, map manager, and support functions.

4.2.3.2 Data preparing

The query process: starts when a user confirmed the selected cluster, then the controller posts a request to the server action *GetIniNode* (in C# controller), and in the callback function of this post, the controller generates a *correctState* object to save the *iniNode*, and adds it to a dictionary. Then for each final state of each arc coming out from *iniNode*, the controller posts requests to the server action *GetStates* (in C# controller), and then recursively does the same for each state in the callback. The received nodes are stored in a list of *LastNodes* for the next query. Each node has a variable *through* to mark if this node has already been queried before or not.

The data stores: in front end the data is stored as objects, and the data model was defined according to the data that come from the back-end and the functions that it should have. After this, the objects are added to its dictionary depending on its type. If an element is *Node*, the controller calculates its coordinates using its *Column* and

Area passed from the back-end. In addition, the *Node* is added to a two-dimensional array *map,* which is used by *Map Manager* to manage the current screen, its index in the map is calculated from its coordinates. In addition, the key of the *Node* is added to a list for the association of searching a *Node*.

The attributes of each *Node* are obtained from the JSON object received from the C# controller.

Asynchronous request problem: As the request to the back-end is asynchronous, the problem is how we may know when the map is loaded completely. The adopted solution consists in recording the number of requests sent, and the node that had been already queried. Once every node has been marked in "through" as true, and every request has received the response, then the loading is considered complete.

4.2.3.3 Initial call

The initial call is the function that prepares a clean SVG document and obtains its root. It also activates the event listeners for user's actions.

When the controller starts to query the cluster data, it verifies if all the nodes in the list *LastNodes* have been already retrieved or not, if they have, it means that the loading is completed and all data have been stored in the browser storage. Then the controller runs the initial call.

Create SVG content

Before creating the content, the initial call first removes the old SVG graphic, clears all the object storage, and resets all counters to initial state.

Then it creates a SVG element with two things:

<defs> tag: This is a child element under SVG element, to predefine reusable elements in SVG images. In our case, we need to define an arrow marker, so that each time we draw an arrow we can reuse the marker instead of drawing it every time.

Background: The background is not required at the beginning, because it is used as a helper for user action *Move Screen*. For the user action *Move Screen*, instead of listening to the entire SVG, we decided to only listening to a background. The reason was to solve the conflict between *Move Screen* and *Move Node*, if we listen to the entire SVG we could not tell if the user wants to move the screen or only move a node in it.

The background is not originally supported by SVG, but we used a *<rect>* tag to simulate it. We generated a rectangle with the same size of the SVG container, and when the user moves *ViewBox,* the background moves at the same time.

Set listeners for user actions on graphic

When initializing the graphic, we added some listeners to the graphic in the SVG element and the background to handle the following user's actions:

- **Move Screen:** As mentioned before, to avoid the conflict with *Move Node*, we added a listener in the background of mouse down, that will enable a listener of mouse move and a listener of mouse up. With the mouse move event, the handler will receive the new mouse position *ClientX, ClientY* and will calculate the offset using the current mouse position. Then it will make the same changes in the SVG *ViewBox* and call the *Map Manager* to apply the same movement (*Move Right/Left/Up/Down*). Finally, the mouse up listener will disable the mouse move and itself.
- **Zoom:** To do this, we added a listener in SVG element that will handle the mouse wheel event. When zooming in, it will check if the scale is within range, if is, it will reduce the scale by half, then set the SVG *ViewBox* size to that scale and call the *Map Manager* to *Zoom In*. Likewise, when zooming out, it will check the scale range, and if it is within range, it will double the scale, set the SVG *ViewBox* size properly, and then call the *Map Manager* to *Zoom Out*.

4.2.3.4 Map manager

The map manager is a class defined to manage the SVG graphic. An instance of this is created at the beginning. The task of this manager is to draw the elements inside the ViewBox and also outside the ViewBox, and when the ViewBox changes, the manager draws new elements and erases too-far-away elements.

The members of the class MapManager are the following ones:

- **Map**: this attribute is a two-dimensional array that contains all the nodes of the extended automaton associated with the selected cluster.
- **Current screen:** this attribute is an object that stores the current cache range. We set it as the screen size of ViewBox.
- **Current screen cache:** this attribute is calculated from the *Current Screen,* and the cache size is set with one screen more in each direction.
- **Generate Map:** after the loading of the map is over, the controller calls this method to generate the current map. So, the manager takes the *map*, traverses the part of the current screen cache, and calls the function draw for each element in it.
- **Move Right:** this function traverses the next column by the right side of the current screen cache in *map* and calls the function draw of each existing *Node*. Then it traverses the last column by the left side of current screen cache in *map* and calls the function undraw of each existing *Node* in this column.
- **Move Left:** this function traverses the next column by the left side of the current screen cache in *map* and calls the function draw of each existing *Node*. Then it traverses the last column by the right side of current screen cache in *map* and calls the function undraw of each existing *Node* in this column.
- **Move Up:** this function traverses the next row by the topside of the current screen cache in *map* and calls the function draw of each existing *Node*. Then it traverses the last row by the bottom side of current screen cache in *map* and calls the function undraw of each existing *Node* in this column.

- **Move Down:** this function traverses the next row by the bottom side of the current screen cache in *map* and calls the function draw of each existing *Node*. Then it traverses the last row by the topside of current screen cache in *map* and calls the function undraw of each existing *Node* in this column.
- **Zoom In:** this function traverses all the *Nodes* in the area just hidden and calls their function undraw to remove them from the SVG.
- **Zoom Out:** this function traverses all the *Nodes* in the area just made visible and calls their function draw.

4.2.3.5 Support functions

There are some other support functions in the controller, which are described below:

User actions:

- SelectCluster: This function supports the selection of the cluster map. Each time the user chooses a domain, this function queries the back end for the list of clusters in this domain, and shows them in the cluster selection options.
- MouseOverNodes: This is the handler of the *Node* mouse over event. It changes the opacity of the *Node* and displays its name.
- MouseOutNodes: This is the handler of the *Node* mouse out event. It reverts its opacity and hides its name.
- MouseOverNodesDetail: This is the handler of the *Node Detail Icon* mouse over event. It changes its opacity and displays the name of its *Node*.
- MouseOutNodesDetail: This is the handler of *Node Detail Icon* mouse out event. It reverts its opacity and hides the name of its *Node*.
- MouseDownNodes: This is the handler of *Node* mouse down event. In the mouse down event, this handler adds two listeners on the Node, one listens to the mouse move and records the change of the mouse position, then makes the same change on the *Node's* coordinates, and calls the function redraw of the *Node* to display the change. The other listens to the mouse up,

which deselects the *Node* and removes these two listeners.

- MouseOverArrows: This is the handler of *Arrow* mouse over event. It changes the opacity and the color of the *Arrow*, and displays the name of it, the name of the *Node* it comes out, and the name of the *Node* it goes to.
- MouseOutArrows: This is the handler of *Arrow* mouse out event. It reverts the opacity and the color of the *Arrow*, and hides the name of it, the name of the *Node* it comes out, and the name of the *Node* it goes to.
- WindowToSVG: This is a support function that calculates the relative position of the current mouse to the SVG container.
- ShowNodeDetail: This is the handler of the *Node Detail Icon* click on event. It calls the function *showDetail* of the *Node* clicked.
- GoToNode: This is the function to search a certain Node. It verifies if the *Node* exists or not, and moves the *ViewBox* to the existing *Node*, and shows the details of this *Node*. Alerts if the *Node* is not found.
- ShowCurrentNode: This is the hander of the *Node* click on event. It shows the details of the *Node* in the *currentNode* container.
- ShowCurrentArrow: This is the hander of the *Arrow* click on event. It shows the details of the *Node* in the *currentArrow* container

SVG creation:

- CreateSVG: This is a support function that creates a SVG element with an id. Before creating the SVG, it checks the current browser version, and then creates the SVG due to different browser (especially IE).
- GetSVGDocument: This is a support function that returns the SVG document, checks the browser version and gets the document due to different browser (especially IE).
- GetSVGRoot: This is a support function that returns the SVG element root, checks the browser version and gets the document due to different browser (especially IE).
- CreateSVGtspan: Current SVG approach does not support text word wrap.

Text tag or a tspan tag will simply draw the whole sentence in a single line. This function is an approach to achieve the word wrap in SVG. It takes a father text tag and a string as input, calculates the words length in the string, wraps it into tspans, and then appends the tspan under the father text tag, then returns the number of tspans made.

Map loading:

- CheckForm: This is the support function to check if the input form of the domain and cluster are in correct format.
- SelectRowbyArea: This is the support function for data preparing, calculates the row of the *Node* based on its *Area* attribute.
- SelectFrequencybyArea: Since we want to display the frequency of the Area *RelevantErrors* in percentage, this function returns the percentage of frequencies if the Area is *RelevantErrors.*
- OrderCorrectFlow: Since the correct flow area needs to be ordered by its regular sequence, this function will be called after a *Node* in correct flow is received, it will reorder the CorrectFlow to its correct sequence based the *Nodes'* keys
- MapLoader: This is a class to manage the loader, an instance of this class is created at the beginning, and is called during the map-loading phase. It contains three public functions:
    - OnLoading: called at the beginning of the map-loading, and generates a process bar
    - Processing: called on each $post iteration done, and advances the process bar.
    - Done: called when the loading phase is done. Advances the process bar to full, and then removes it.

Initiation:

- ClearAllDictionary: This is the function that clears all dictionaries, and is used in the initial call.

- ResetAllCounters: This is the function that clears all counters, and is used in the initial call.

Others:

- Dictionary: As JavaScript does not have the data structure Dictionary or HashMap, we decided to implement a Dictionary structure to store the data. It implements the default functions of a Dictionary class in C#.
- FindNodeByNodeKey: This is a support function that traverses all dictionaries and returns the *Node* object by its key if exists, otherwise returns a null object.

4.2.4 Design of the View

4.2.4.1 Layout

In Microsoft ASP.NET MVC scheme, the page view is generated by the file _ViewStart.cshtml, this file indicates that the layout file is _Layout.cshtml.

In _Layout.cshtml we removed the page footer, and modified the *navbar* options to only show the name of the application, and in the body we show the View, the critical part of code is shown below: …

```
<div class="navbar-header">
    @Html.ActionLink("Student Predictor Viewer", "Index", "Home", new { area
    = "" }, new { @class = "navbar-brand" })
</div>

…
<div class="container body-content">
    @RenderBody()
    <hr />
</div>

…
```

4.2.4.2 Index

Then, we created a View in Index.cshtml. To achieve a better outlook of the page, we

decided to import Bootstrap as the page style. Then the view was divided into two container classes.

The first container contains four parts:

- Figure 19 shows the cluster selector for user to select the cluster.

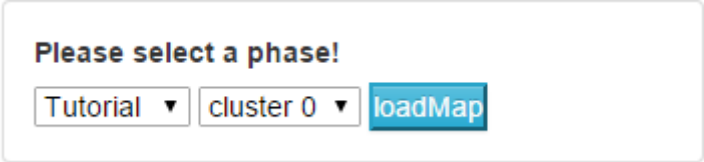


Figure 20 - Cluster Selector preview

- Figure 20 shows the current node container to show the selected node

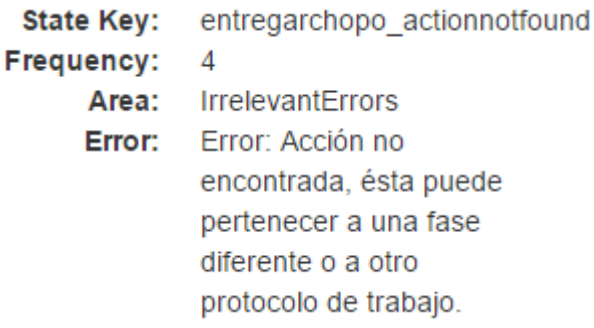


Figure 19 - Current Node preview

detail.

- Figure 21 shows the current arrow container to show the selected arrow detail.

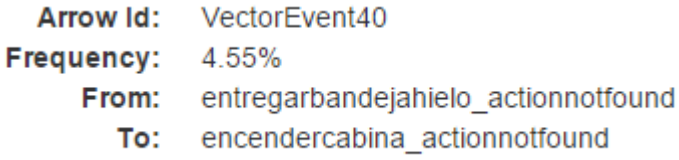


Figure 21 - Current Arrow preview

- Figure 22 shows the loader image: a circle process loader when loading the map.

Figure 22 - Process bar preview

The second container contains two parts:

- Figure 23 shows the search box for node searching, also contains a list of the association options
- Then there is the SVG graphic container, which will initially be blank.

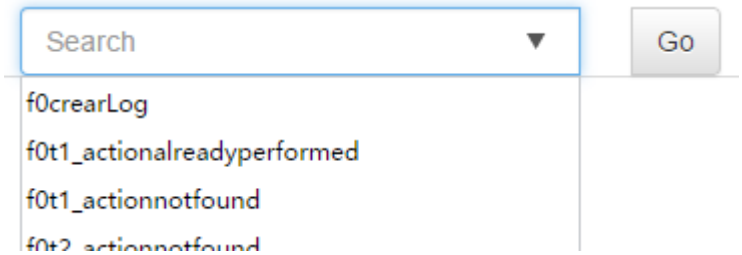


Figure 23 - Search Box preview

Chapter 5

5. Testing Phase

The application testing was done following the agile testing process. Test cases were extracted from the application requirements. Each requirement corresponds to one or more test cases.

Scenario 1

Scenario Number: IDX001

Scenario Name: User loading cluster map

Test case 1

Case Number: IDX001-01

Preconditions: User visits the application through a web browser

Test process&Input data:

User clicks the drop-down list for domain selection and selects one domain. Figure

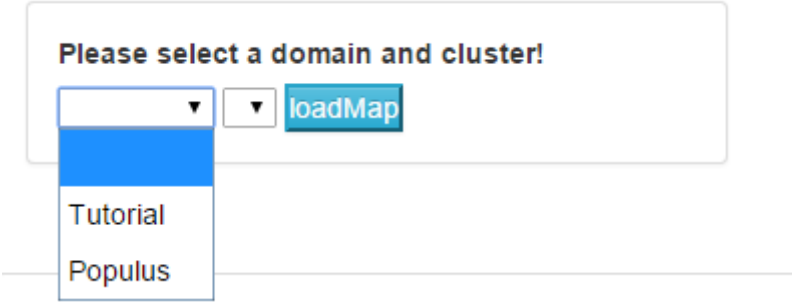


Figure 24 - Test Case IDX001-01

24 shows the test process.

Expected result:

The application returns the clusters within the selected domain.

Actual result:

The application returns the clusters within the selected domain.

Figure 25 shows one possible results of the test case.

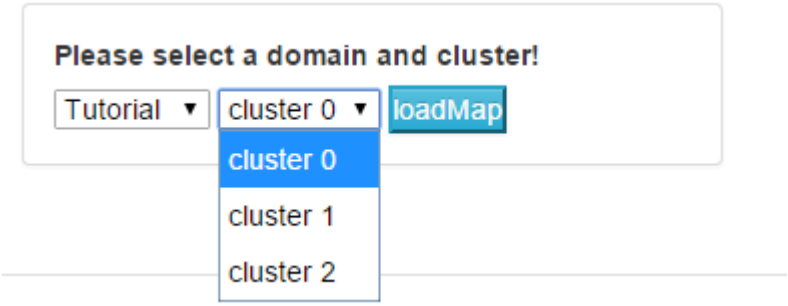


Figure 25 - Test Case IDX001-01

Test case 2

Case Number: IDX001-02

Preconditions: User has selected a domain and a cluster

Test process&Input data:

User clicks the *load map* button, and is waiting for the map to be loaded.

Expected result:

After the user clicked the *load map* button, the application shows a progress bar that represents the loading process. When the loading ends, the progress bar disappears and the map is displayed.

Actual result:

The map was correctly loaded, Figure 26 shows the process bar during the loading, Figure 27 shows the generated map after the loading was completed.

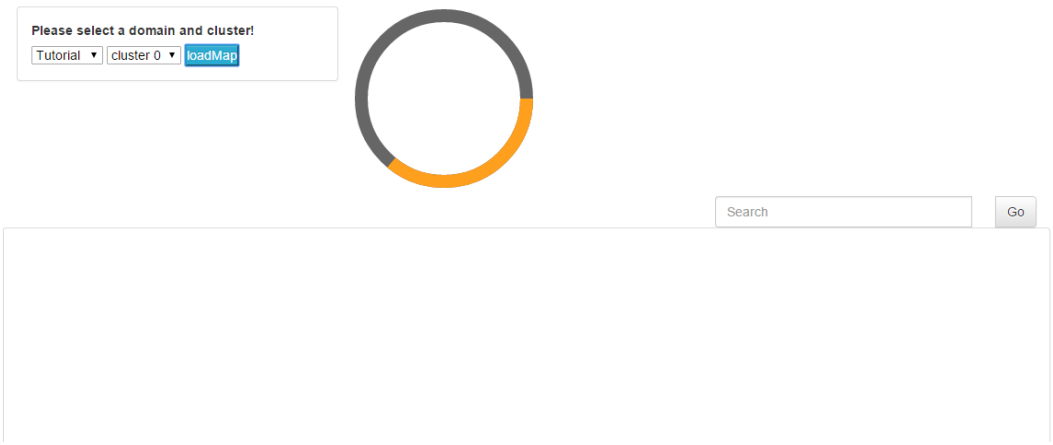


Figure 26 - Test Case IDX001-02

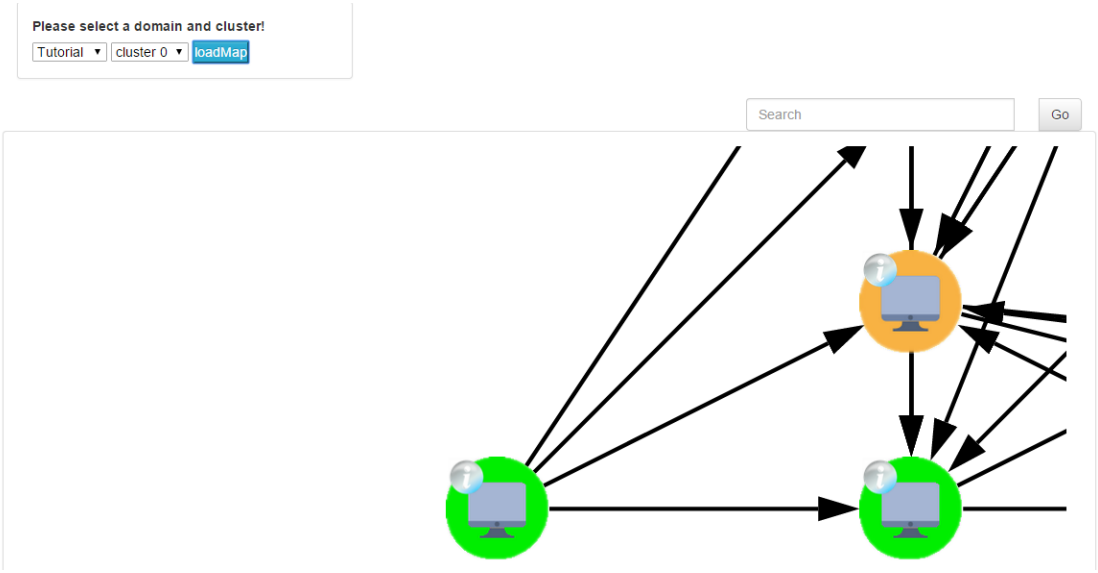


Figure 27 - Test Case IDX001-02

Scenario 2

Scenario Number: IDX002

Scenario Name: User navigates across the cluster map

Test case 1

Case Number: IDX002-01

Preconditions: User has loaded a map

Test process&Input data:

User clicks on the cluster map, dragging the map to up, down, left, and right within the graphic area.

Expected result:

When user drags the map in the graphic area, the cursor changes to move style, and the graphic is translated in the same direction as the mouse moves.

Actual result:

The graphic interacts correctly with the user's actions, Figure 28 shows a graphic when the user clicked on the map and dragged it.

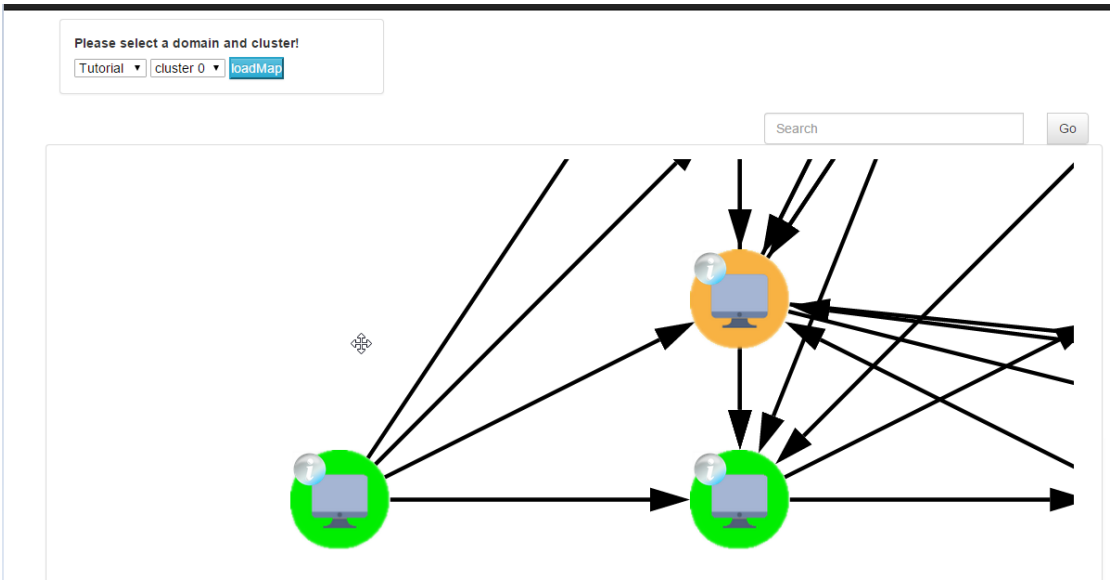


Figure 28 - Test Case IDX002-01

Test case 2

Case Number: IDX002-02

Preconditions: User has loaded a map

Test process&Input data:

User uses the mouse scroll wheel when the mouse is over the graphic area.

Expected result:

The map should zoom in when the user wheels up the mouse scroll wheel, and the map should zoom out when the user wheels down the mouse scroll wheel with the

pointer over the graphic area.

Actual result:

The map reacts as expected, and zoom actions corresponds to mouse events. Figure 29 shows a graphic zoomed in and Figure 30 a graphic zoomed out.

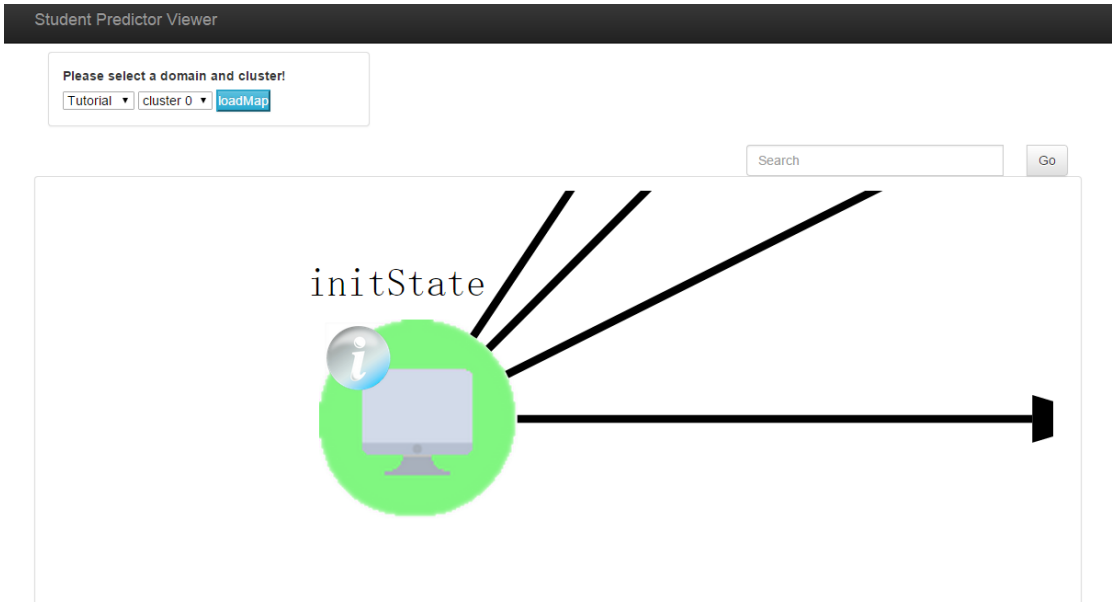


Figure 29 - Test Case IDX002-02

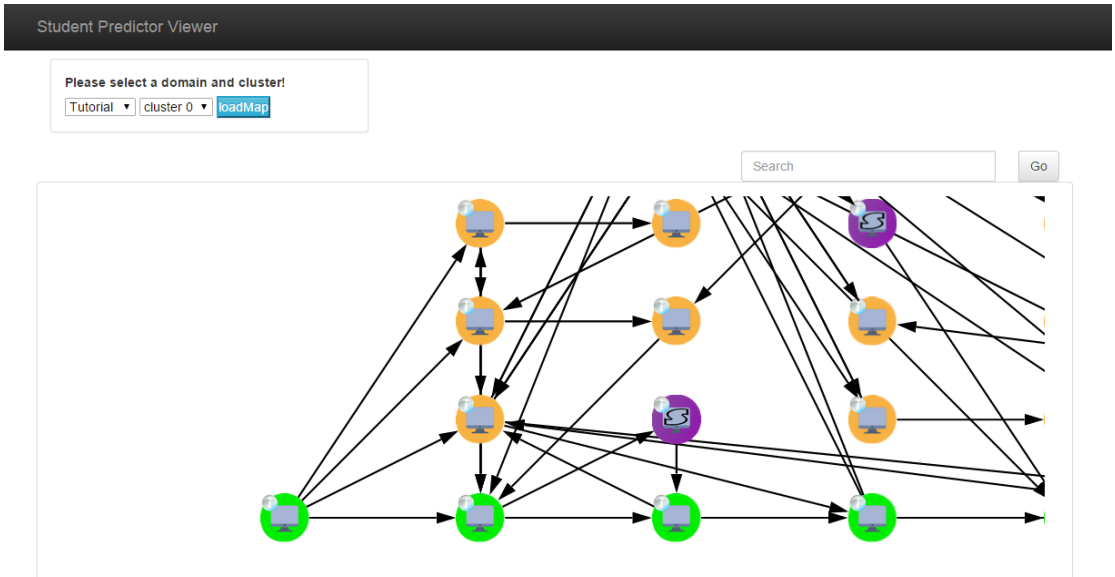


Figure 30 - Test Case IDX002-02

Scenario 3

Scenario Number: IDX003

Scenario Name: User visualizes the detail of a *Node* or an *Arrow*

Test case 1

Case Number: IDX003-01

Preconditions: User has loaded a map

Test process&Input data:

User points at a *Node*, and clicks on the *Node* or on the *Detail Icon* of the *Node* to show a detail, or to hide the detail when it is being shown.

Expected result:

The application should highlight the *Node* and shows the *Node's* name when the mouse is over it. Then, when user clicks on the *Node*, the application should show the *Node's* detail out of the graphic. And when the *Detail Icon* is clicked, a box under the *Node* with the *Node's* detail information should be shown, otherwise if the detail box was already shown, the application should hide this box.

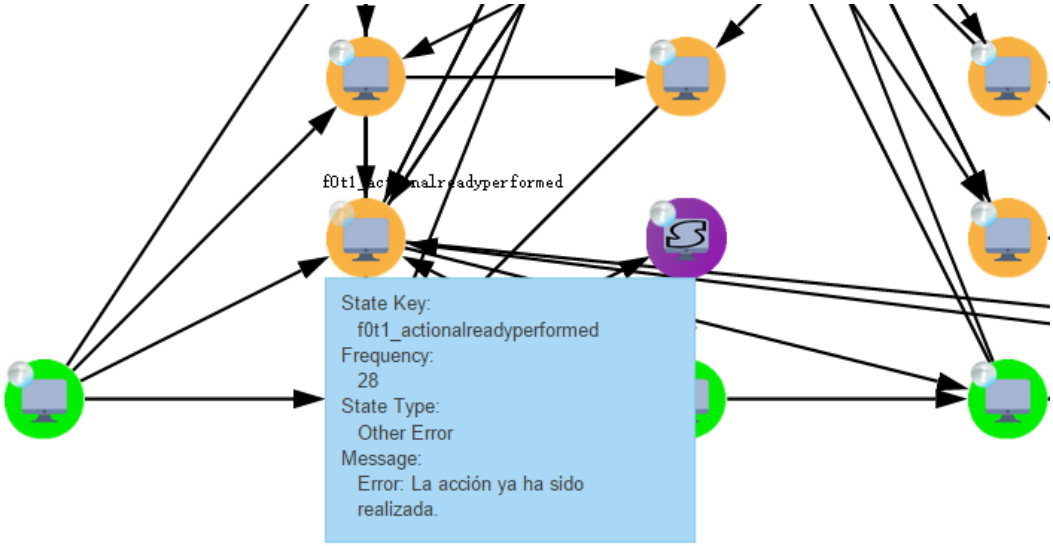


Figure 31 - Test Case IDX003-01

Actual result:

The application reacts as expected. Figure 31 shows the detail box shown by the application. Figure 32 shows the detail of a *Node* shown out of the graphic.

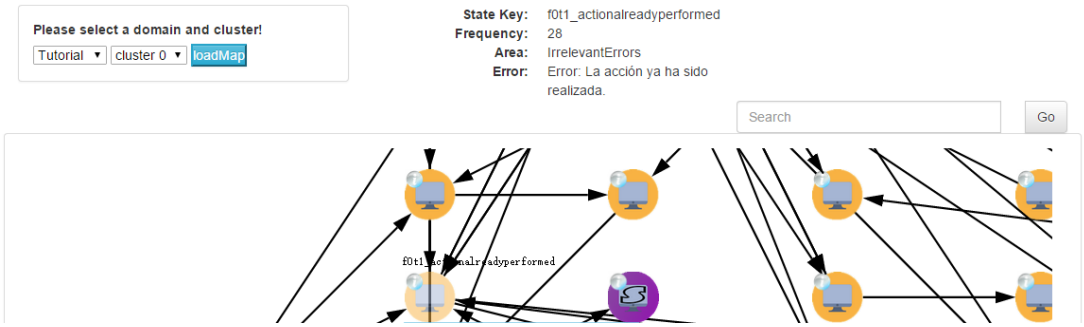


Figure 32- Test Case IDX003-01

Test case 2

Case Number: IDX003-02

Preconditions: User has loaded a map

Test process&Input data: User points at an *Arrow*, and click on it.

Expected result: The application should highlight the *Arrow* and show its detail aside de mouse when mouse is moved over it. If user clicks the *Arrow*, its details should be

shown out of the graphic.

Actual result:

The application reacts as expected. Figure 33 shows the details shown by the application when mouse is moved over an Arrow. Figure 34 shows the detail of an Arrow shown out of the graphic.

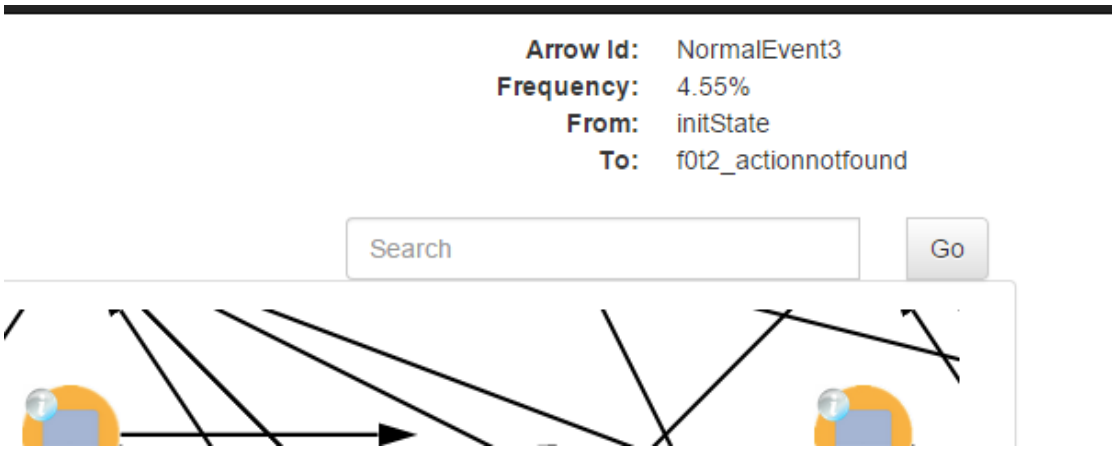


Figure 33- Test Case IDX003-02

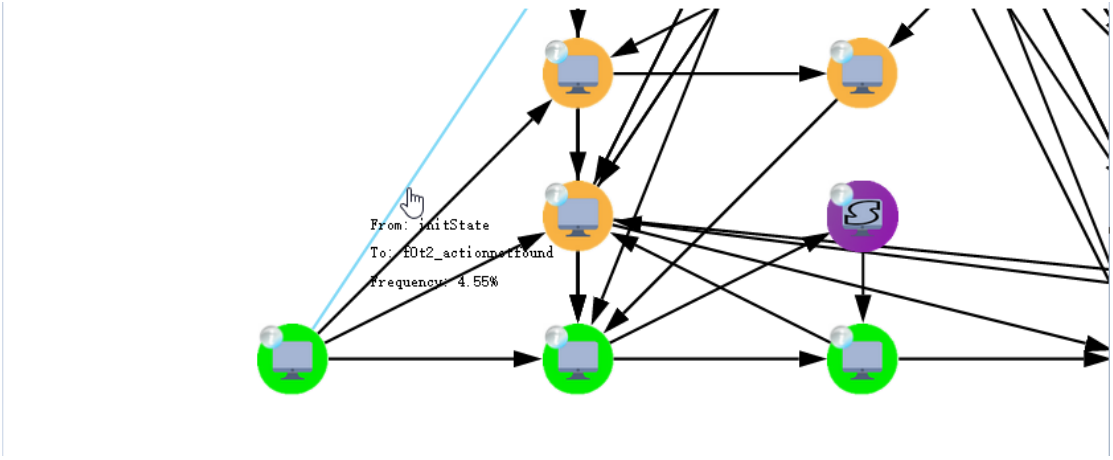


Figure 34- Test Case IDX003-02

Scenario 4

Scenario Number: IDX004

Scenario Name: User searches a certain *Node* in the map

Test case 1

Case Number: IDX004-01

Preconditions: User has loaded a map

Test process&Input data:

User inputs a *Node* name in the search box, and click Go button

Expected result:

When user is typing in the search box, the application should show a list of available matching state keys for user. When user finishes typing and clicks Go button, the application should move the *ViewBox* to the state and show its details, or alerts to the user with an error message if the entered key is not found.

Actual result:

The application behaves as expected. Figure 35 shows the association appeared for the user. Figure 36 shows the result found for a Node name. Figure 37 shows the error message when the *Node* was not found.

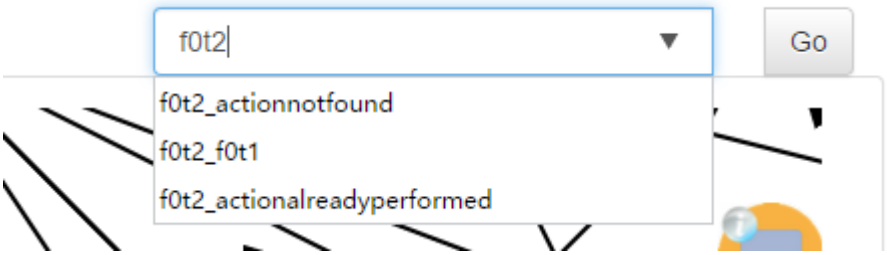


Figure 35 - Test Case IDX004-01

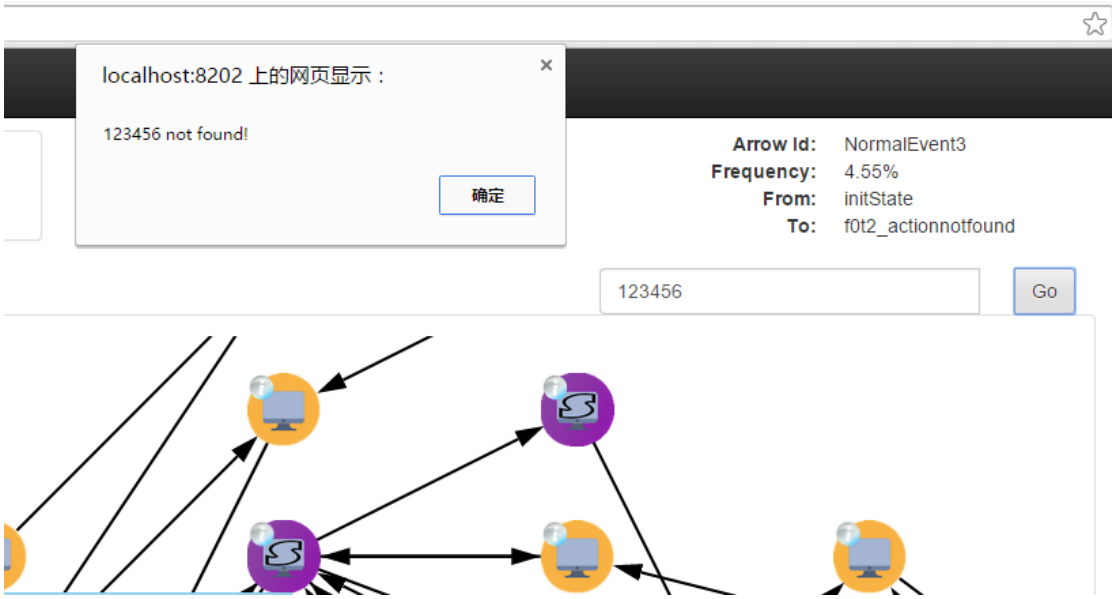


Figure 36- Test Case IDX004-01

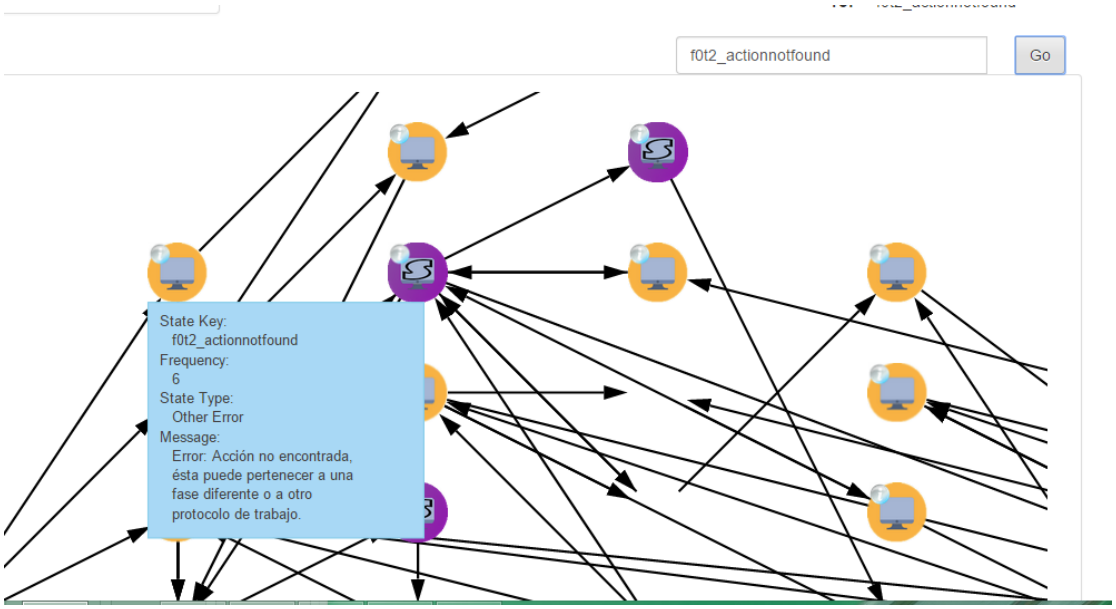


Figure 37 - Test Case IDX004-01

Chapter 6

6. Conclusions and Future Work

6.1 Conclusions

During the implementation of this project, we found that there are still very little documentation on SVG or HTML 5 implementations. Despite of that, we think that they are the incoming trend in web application development.

HTML 5 has fundamentally changed the way of that developers are implementing web applications, from desktop browser to mobile applications. This language and its standards are affecting and will continue to affect a variety of operating platforms. HTML5's rapid growth is worthwhile for all to gain close attention, in fact, in the last years, many companies have entered in this field. HTML 5 will be like traditional Flash, Flex, Silverlight, or Objective-C, and will form its own ecosystem of frameworks and tools. HTML 5 is more likely to emerge in any device than Flash, Flex, Silverlight, and Objective-C.

Since the W3C (World Wide Web Consortium) developed the SVG in 1998, SVG's development has been all the way into a downturn phase. However, since 2014, after HTML 5 included SVG as a web development standard, foreseeably SVG will become one of the top topics in web development. The biggest charm of SVG is the ease of use, besides compared to traditional graphic image formats, the file size of SVG

document is surprisingly small. The born of SVG has a very positive meaning for the web applications that are currently facing various problems, therefore its potential prospects are very broad. It could be expected that with SVG will gradually be applied in applications such as mathematical cartography, dynamic graphics generation, online presentations, graphical interaction and games, and it will increasingly become more popular in Web applications. Moreover, SVG is not limited to these applications, because its scalability characteristics are giving an unlimited space to play for developers.

6.2 Future work

In the end of this project there was an unimplemented requirement, which is expand and shrink branches that go out from a *Node*. The reason of that this requirement was not implemented was because of the bad design of the project at the beginning. As the project followed an agile development process, the initial goal of the project was to show the graphic, and this led to a bad design of the project at the beginning. When we came to realize the problem, we decided to reconstruct the project, but we had not sufficient time to finish this requirement. However, the application was designed so that the adding of this new feature could be done easily.

In order to speed up the application, we propose to pass the map part from the JavaScript controller to the C# controller. In this way, the entire map would be created at the App_start phase, so that the JavaScript controller would only need to query the C# controller the current map content, instead of managing the whole map in the browser cache.

References

[1] Diego Riofrío and Jaime Ramírez. (2015) - A Model for Student Action Prediction in 3D Virtual Environments for Procedural Training. In EDM2015: Proceedings of the 8th International Conference on Educational Data Mining.

[2] Depth understanding of ASP.NET MVC (Chinese)

[3] Webforms vs MVC and Why MVC is better

[4] Top 10 Changes in ASP.NET 5 and MVC 6

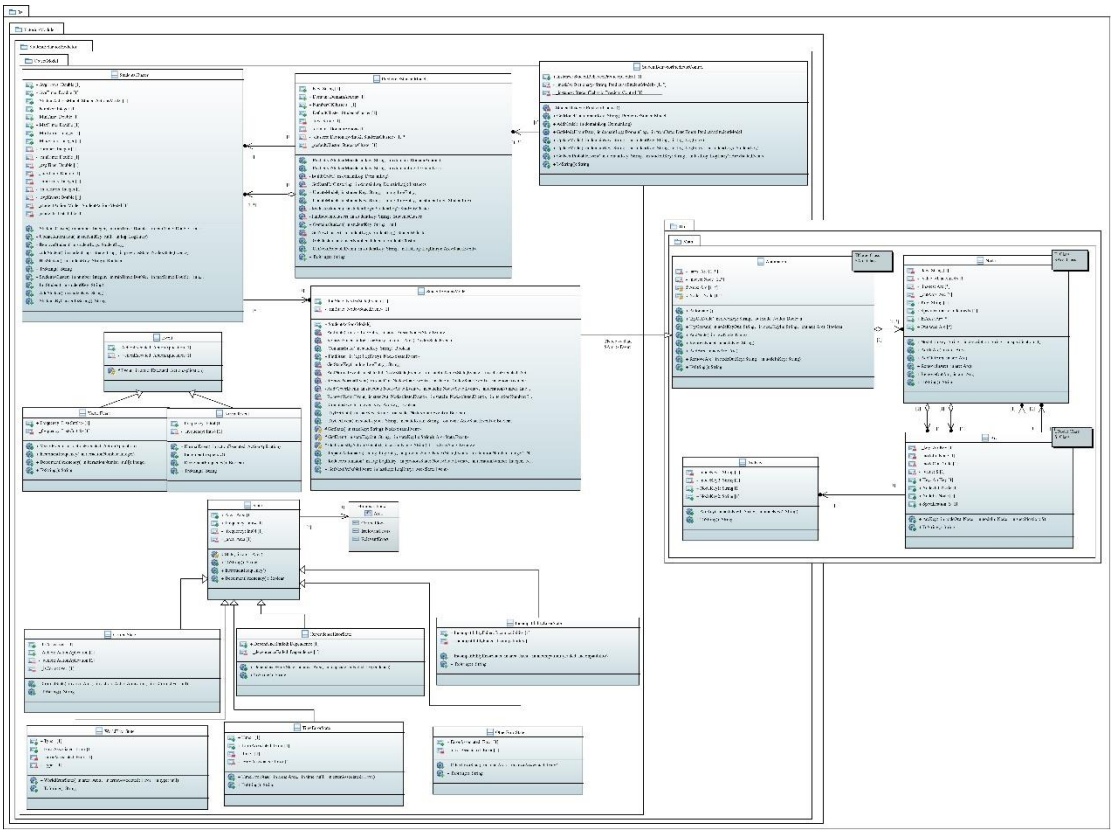
[5] 10 big advantage of HTML5 (Chinese)

[6] Advantages and limitations of JavaScript (Chinese)

[7] Scalable Vector Graphics

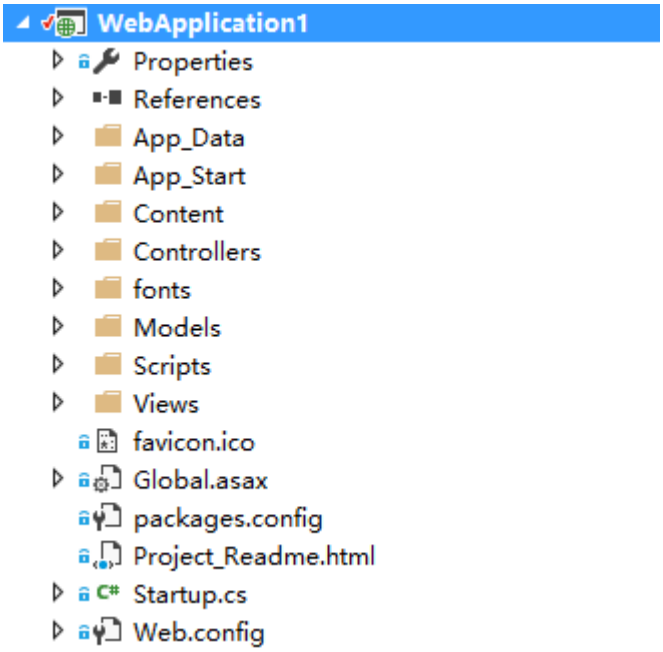
[8] Xamarin

Annex 1: Student Behavior Predictor class diagram



Annex 2: Structure of a .NET MVC structure project

As shown above, the basic functions of each one is:



Properties: General configurations of the solution, including building, debugging and webserver configurations.

References: Project libraries.

App_Data: This folder is the local data storage of the application. It usually contains the data in document form. The folder contents won't be processed by ASP.NET. The folder is the default location for ASP.NET data storage

App_Start: This folder contains classes that shall be run at the beginning of the application.

Content: Contains files for styles.

Controllers: The main storage location for controllers, new controllers are required to be created in this directory.

Fonts: Contains font's style files.

Models: The location for define entities, as well as with the related operation classes

Scripts: JavaScript libraries, already place with commonly used jQuery/jQuery-Ui and some JavaScript libraries.

Views: This part is to place the display section of View.

Global.asax: The entrance of the application, normally shall call classes from App_Start folder.

Package.config: The configuration of the solution defining external packages.

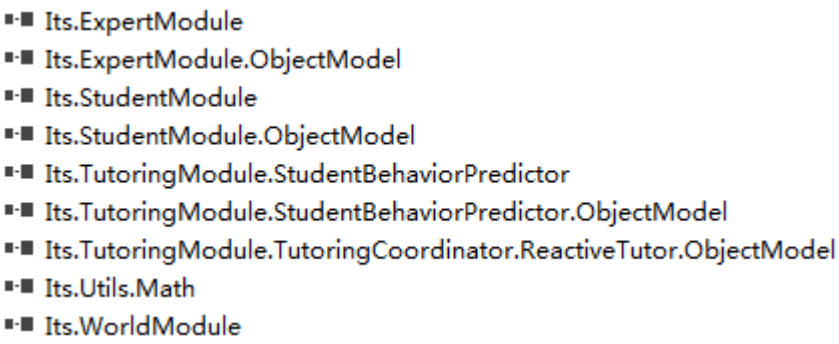
Web.config: General configuration file for the application.

Next, we will explain the role of each of the components outlined in figure X.

Annex 3: Configuration in importing Student Model

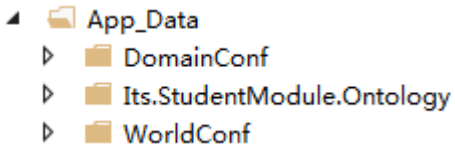
Adding reference

The first step was to import the model into the project. As the project was created in the Tutor Solution under the package of *TutoringModule\StudentBehaviorPredictor*, we started adding only the references paths in the project.



The problem was that IIS Express couldn't obtain the physical local path of the references, as the references were outside the project.

The solution was to add the entire outside references in APP_Data folder, and upload it to the server when running.



Configuration

After adding the reference, some configuration should be modified in web.config in order to run the model

The *ontologyPath, logsPath, domainConfigurationPath, worldConfigurationPath* were settings of paths, and modified to indicate the current packages in APP_Data.

The *domain1, domain2* were names of current exsting modules in Student Model.

Others were settings for create model.

```
<appSettings>
  <add key="initialColumn" value="2" />
  <add key="finalColumn" value="21" />
  <add key="initialRow" value="1" />
  <add key="ontologyPath" value="~/App_Data/Its.StudentModule.Ontology/" />
  <add key="logsPath" value="~/App_Data/Its.StudentModule.Ontology/Logs/" />
  <add key="domainConfigurationPath" value="~/App_Data/DomainConf/" />
  <add key="worldConfigurationPath" value="~/App_Data/WorldConf/" />
  <add key="TimePenalization" value="86400" />
  <add key="NoTutoringNoBlockErrorPenalization" value="2" />
  <add key="TutoringBlockErrorPenalization" value="5" />
  <add key="TutoringNoBlockErrorPenalization" value="10" />
  <add key="domain1" value="Tutorial" />
  <add key="domain2" value="Populus" />
</appSettings>
```

Annex 4: JSON data formation

```
VectorEvent = {
    NodeKey1: String,
    NodeKey2: String,
    Frequency: Array,
    Element: String
}
```

```
NormalEvent = {
    NodeKey1: String,
    NodeKey2: String,
    Frequency: Int,
    Element: String
}
```

```
CorrectState = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    Name: String,
    Columns: Int,
    StateType:String
}
```

```
WorldErrorState = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    Message: String,
    Columns: Int,
    StateType: String
}
```

```
TimeErrorState = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    Message: String,
    Time: Int,
    Columns: Int,
    StateType: String
}
```

```
IncompatibilityErrorState = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    Message: String,
    Action: String,
    Columns: Int,
    StateType: String
}
```

```
OtherErrorState = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    Message: String,
    Columns: Int,
    StateType: String
}
```

```
SimpleDependence = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    Message: String,
    Action: String,
    Columns: Int,
    StateType: String
}
```

```
iniNode = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    TotalStudent: Int
}
```

```
SeqComplexDependence = {
    Key: String,
    Area: String,
    Element: String,
    Frequency: Int,
    Message: String,
    ComplexDependenceKey: String,
    Columns: Int,
    StateType: String
}
```